Servlet and JSP Programming

with IBM WebSphere Studio and VisualAge for Java

Teach yourself servlet and JSP programming techniques

Develop and test with WebSphere Studio and VisualAge for Java

Deploy to WebSphere Application Server

Ueli Wahli
Mitch Fielding
Gareth Mackown
Deborah Shaddon
Gert Hekkenberg

Redbooks

ibm.com/redbooks

SG24-5755-00

International Technical Support Organization

Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java

May 2000

First Edition (May 2000)

This edition applies to Version 3.02 of WebSphere Application Server, WebSphere Studio, and VisualAge for Java for use with the Windows NT Operating System. Many of the concepts also apply to these products running on AIX, UNIX, and OS/2 Operating Systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. OWR Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This IBM Redbook provides you with sufficient information to effectively use the IBM WebSphere and VisualAge for Java environments to create, manage and deploy Web-based applications using methodologies centered around servlet, JavaServer Pages, and JavaBean architectures.

In Part 1 we describe the products used in our environment and provide instruction on product installation and configuration. Following this, we cover servlet and JSP programming, which provide you with both a theoretical and practical understanding of these components, together with working examples of the concepts described. For execution of the sample code, we provide information on configuring the WebSphere Application Server and deploying and running the sample Web applications in WebSphere. Using the knowledge developed in these chapters, we then provide detailed information on the development environments offered by VisualAge for Java and WebSphere Studio. These chapters assist you in using the features offered by these tools, such as integrated debugging, the WebSphere Test Environment, Studio Wizards, and publishing of Web site resources. We also describe how Rational's ClearCase product can be integrated with our environment for Software Configuration Management.

In Part 2 we describe the Pattern Development Kit sample application, including installation, configuration, and operation. We also discuss the application's use of Patterns for e-business, which presents information on some of the design decisions employed when creating the application.

This IBM Redbook is intended to be read by anyone who requires both introductory and detailed information on software development in the WebSphere environment using servlets and JavaServer Pages. We assume that you have a good understanding of Java and some knowledge of HTML.

Sample code on the Internet

The sample code for this redbook is available as the 5755samp.zip and 5755pdk.zip files on the ITSO redbooks home page on the Internet:

`ftp://www.redbooks.ibm.com/redbooks/SG245755/`

Download the sample code and read Appendix C, "Using the additional material" on page 417.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Ueli Wahli is a Consultant I/T Specialist at the IBM International Technical Support Organization in San Jose, California. Before joining the ITSO 16 years ago, Ueli worked in technical support at IBM Switzerland. He writes extensively and teaches IBM classes worldwide on application development, object technology, VisualAge products, data dictionaries, and library management. Ueli holds a degree in Mathematics from the Swiss Federal Institute of Technology. His e-mail address is *wahli@us.ibm.com*.

Mitch Fielding is an e-business Specialist working with FishTech & Partners—an IBM Business Partner based in Sydney, Australia. He has 10 years experience in software development and consulting in both private and government sectors. He is currently working on new Internet-based products centered around WebSphere technology and developed with VisualAge for Java and DB2. His e-mail address is *mfieldin@fishtech.com.au*.

Gareth Mackown is an Advisory I/T Specialist working within e-business Services in Hursley, England. He has worked at IBM for nearly 5 years, predominantly developing and consulting, though occasionally teaching. His areas of expertise center around Java and include VisualAge for Java, WebSphere and object technology. Gareth holds a Joint Honors degree in Mathematics and Computer Science from Durham University. His e-mail address is *gareth_mackown@uk.ibm.com*.

Deborah Shaddon is a Senior I/T Specialist from IBM Global Application Delivery in Chicago, Illinois. She has over 12 years of application development and architecture experience primarily in the Banking and Finance sector. Her current area of expertise includes developing custom e-business solutions for IBM customers, using a variety of technologies, including WebSphere, VisualAge for Java, and Lotus Domino. Deborah holds a degree in Business Information Systems from Bradley University, Peoria, Illinois, and is currently pursuing a Masters in Software Engineering from DePaul University, Chicago, Illinois. Her e-mail address is *dmshadd@us.ibm.com*.

Gert Hekkenberg is a Senior I/T Specialist from IBM Software Group EMEA region North, based in Amsterdam, The Netherlands. He has over 15 years of application enabling experience with a special focus on the broader Software Configuration Management area. He is currently working as Technical Sales Consultant designing E2E application development solutions for large customers. He has written extensively on application development and SCM in various redbooks over the years and was involved developing various ITSO workshops as well. Gert holds a Masters degree in Business Information Systems from Erasmus University, Rotterdam, The Netherlands and a Bachelors degree in economics from Vrije Universiteit, Amsterdam, Netherlands. His e-mail address is *hekkenberg@nl.ibm.com*.

Thanks to the following people for their invaluable contributions to this project:

Pat McCarthy, Joaquin Picon, and Markus Muetschard, IBM ITSO San Jose, for their ongoing support in all aspects of application development and redbook publishing.

Sheldon Wosnick, IBM Toronto, Canada, for helping with servlet development techniques and the configuration of the VisualAge for Java WebSphere Test Environment.

Jonathan Adams, IBM UK, for leading the effort of producing the Patterns for e-business.

The team that produced the Pattern Development Kit described in Part 2:

- ❑Anthony Griffin— IBM Hursley, Pattern Development Kit
- ❑Rob Veck—Advanced Solutions Group, IBM Hursley, Concept of the e-Business Solution Kit
- ❑Joe Parman and Dave Mulley—Advanced Solutions Group, IBM Hursley, Expert Install & Packaging
- ❑Mark Campbell and Robert James—Advanced Solutions Group, IBM Hursley, Graphic Design IBM Hursley

Chris Gerken, IBM Raleigh, US, for the utility JSP.

The IBM WebSphere Application Server, WebSphere Studio, and VisualAge for Java development teams.

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- ❏Fax the evaluation form found in “IBM Redbooks review” on page 447 to the fax number shown on the form.
- ❏Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- ❏Send your comments in an Internet note to `redbook@us.ibm.com`

SPONSORSHIP PROMOTION

IBM Software Accelerated Value.....

takes your IBM technology end to end

Contact us at: software_accelerated_value@us.ibm.com

IBM
Making sense of the world's data
IBM and the IBM logo are trademarks of the International Business Machines Corp., registered in many jurisdictions worldwide.

WebSphere. software

www.ibm.com/software/support/acceleratedvalue

Part 1 Web application development

In this Part we describe general techniques for servlet and JSP programming. We then explain in detail how to run servlets and JSPs in WebSphere Application Server, how to develop and test them in VisualAge for Java, and how to use WebSphere Studio for development and publishing.

We do not describe Enterprise JavaBeans. For more information on how to develop and test Enterprise JavaBeans, please refer to the *Servlet/JSP/EJB Design and Implementation Guide*, SG24-5754.

1 Environment overview

This chapter provides a schematic overview and description of the primary functional areas addressed in this book.

The four areas presented are:

- ❑WebSphere execution environment
- ❑VisualAge for Java development environment
- ❑WebSphere Studio development environment
- ❑VisualAge for Java and WebSphere Studio interactions

The diagrams presented in this chapter are high-level diagrams aimed at illustrating the major components and interactions within each functional area.

WebSphere execution environment

The execution environment used when writing this book and its associated code is based on the diagram shown in Figure 1. The primary components of the environment are:

❑WebSphere Application Server

❑IBM HTTP Server

❑DB2

Some secondary components shown in the execution environment are:

❑Classes and HTML/JSP files

There are many examples throughout this book of servlets, JavaBeans and HTML/JSP pages used by the application server and Web server.

❑IBM SecureWay Directory

IBM SecureWay Directory provides LDAP user authentication for the Patterns Development Kit examples presented in Part 2 of this book.

❑Enterprise Data

Connectors to enterprise data are not covered within this book, however we have depicted this in the execution environment to show that it supports connections to a variety of enterprise data sources. The Patterns Development Kit provides examples for enterprise connectors to CICS and MQSeries, however, they are not discussed in this book.

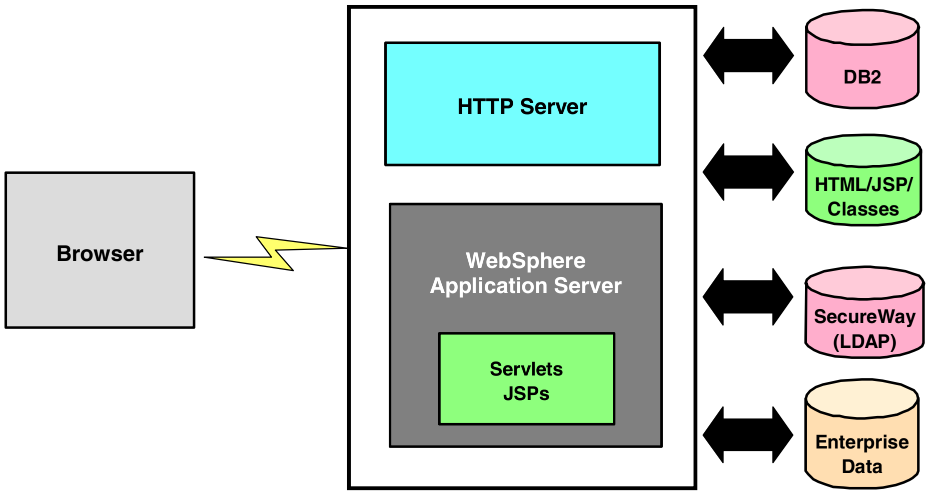


Figure 1. WebSphere execution environment

VisualAge for Java development environment

VisualAge for Java provides extensive functionality across the entire development life-cycle and includes tools for Java code editing and debugging, JavaServer Page debugging, and the WebSphere Test Environment. The development environment is shown in Figure 2.

VisualAge for Java also includes a repository that stores project source and compiled code, and an import/export facility that enables interaction with the file system.

One of the most important features of VisualAge for Java is the WebSphere Test Environment. This feature provides application and Web server environments on a development machine, enabling you to test and debug the resources of a Web site locally. This environment provides much of the functionality of a full application server, including access to services such as LDAP and enterprise resources.

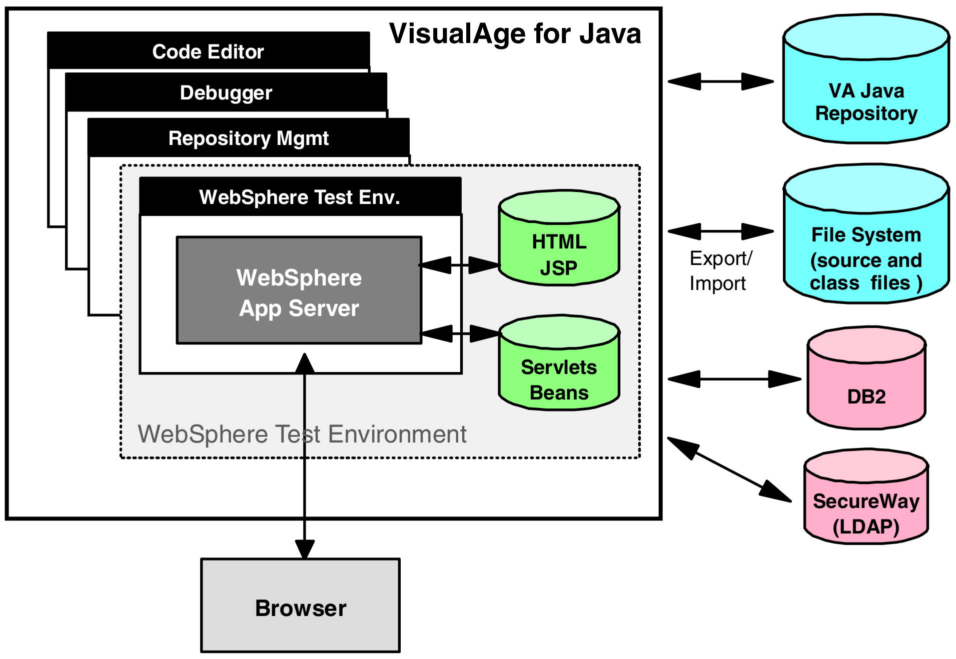


Figure 2. VisualAge for Java development environment

WebSphere Studio development environment

WebSphere Studio is used to develop, manage and deploy the resources for a Web site. Figure 3 shows the primary features and interactions of WebSphere Studio.

WebSphere Studio maintains project files in the file system and provides support for team development and version control tools. The deployment features of WebSphere Studio enable you to configure the projects to deploy to a number of locations, such as the WebSphere Application Server or the WebSphere Test Environment of VisualAge for Java.

WebSphere Studio also contains a number of wizards that guide you through tasks such as SQL statement generation and creation of Web pages to interact with databases and JavaBeans. You can also use the WebSphere Studio Page Designer to edit these generated pages, or create your own HTML and JSP pages.

Any Java source code within WebSphere Studio can be compiled using the supplied Java compiler.

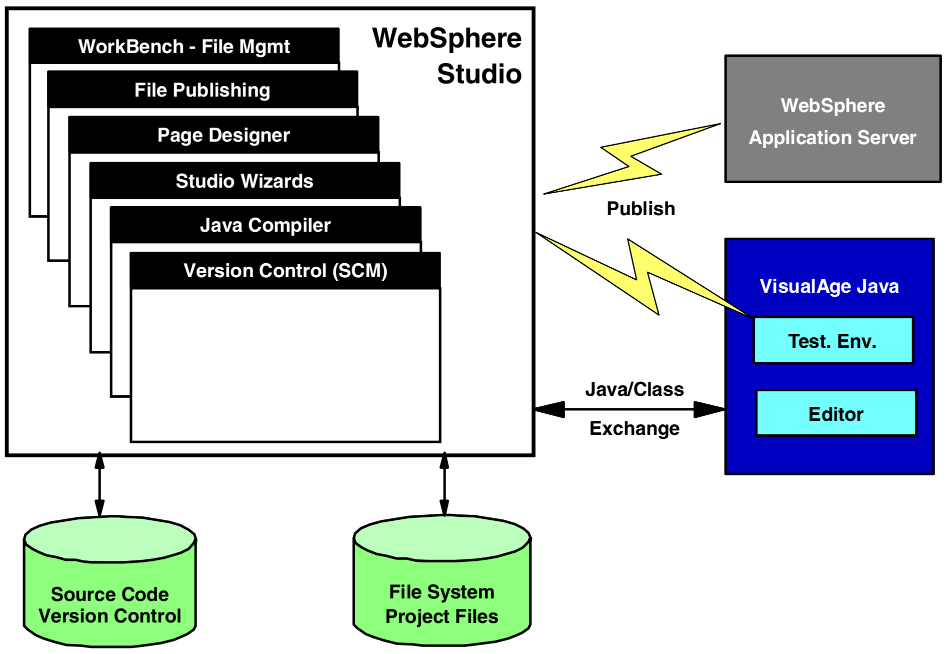


Figure 3. WebSphere Studio environment

VisualAge for Java and WebSphere Studio interactions

An invaluable feature of WebSphere Studio is its ability to interact with VisualAge for Java. Using this feature, enabled through the Toolserver API, you can update code from, and send code to, the VisualAge for Java development environment (Figure 4).

When classes are imported into WebSphere Studio from VisualAge for Java, they are extracted from the VisualAge for Java repository and converted into files. These files are stored in the file system structure used by WebSphere Studio and can subsequently be published to the application server.

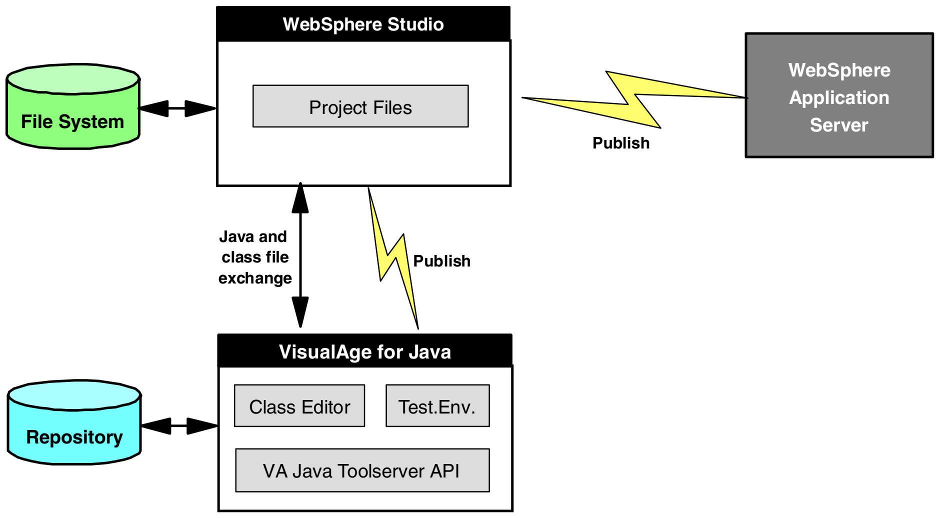


Figure 4. WebSphere Studio and VisualAge for Java interaction

Complete product environment

The complete development and execution environment is shown in Figure 5.

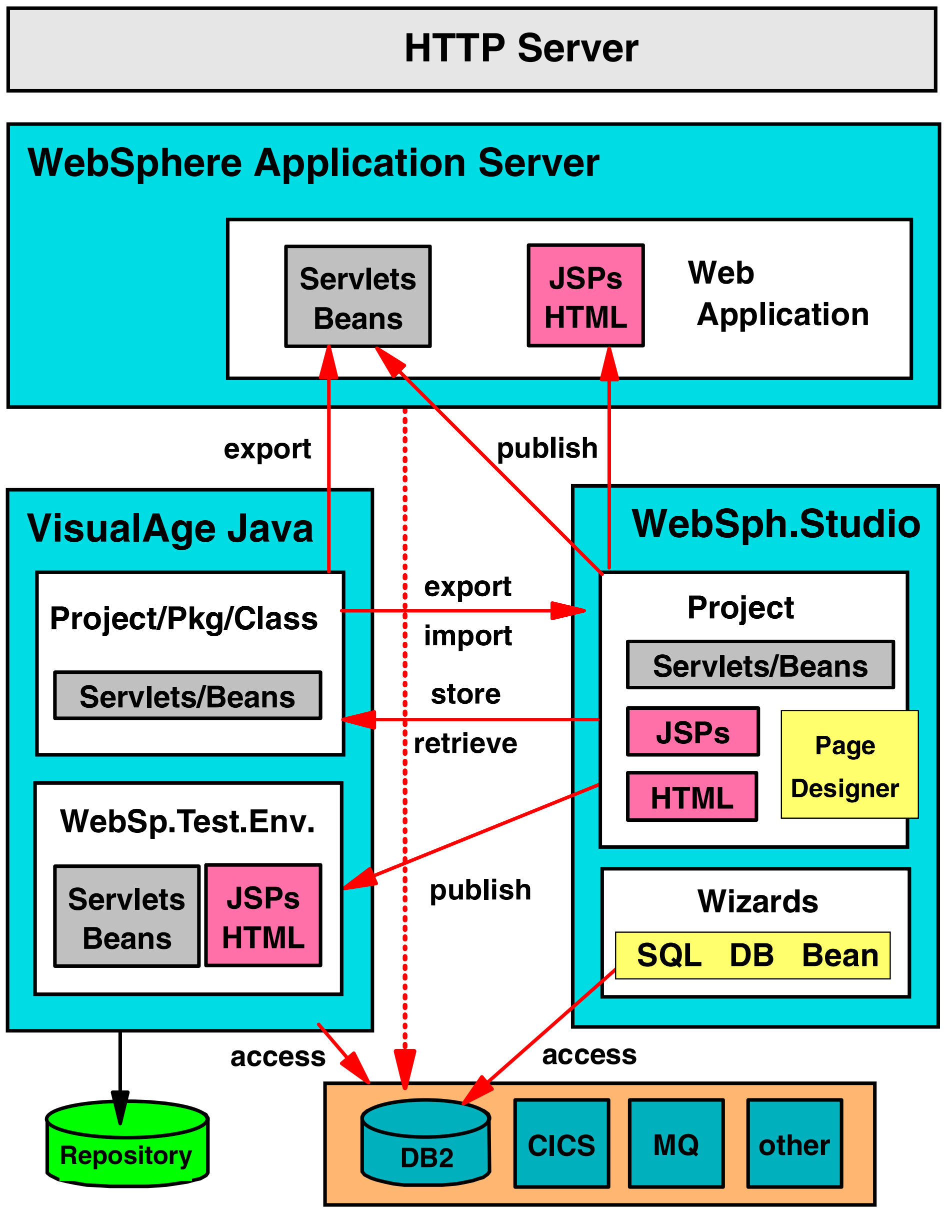


Figure 5. Environment for Web application development and execution

Here is a short description of the major components:

- The HTTP Server serves static HTML pages to browsers.
- The WebSphere Application Server plugs into the HTTP Server for dynamic content generated by servlets and JavaServer Pages (JSP).

  The Application Server supports the concept of a Web application that represents a grouping of servlets, JSPs, and their related resources. Managing these elements as a unit allows you to stop and start servlets in a single step. You can also define a separate document root and class path at the Web application level, thus allowing you to keep different Web applications separated in the file system. Please refer to “Web application” on page 131 for more information on Web applications.
- VisualAge for Java is the product for development and testing of Java applications, applets, servlets, JavaBeans, and Enterprise JavaBeans. It also includes the WebSphere Test Environment, which can be used to test Web interactions involving HTML files, servlets, JSPs, and JavaBeans.

  VisualAge for Java can export Java and class files to WebSphere Application Server and WebSphere Studio.

  VisualAge for Java runs in a team environment with a repository for central storage of the code of many developers.
- WebSphere Studio provides a development environment for HTML files and JSPs. It also provides wizards that generate skeleton Web pages, servlets, and JSPs for database and JavaBean access.

  The Page Designer tool is provided to edit static HTML pages and JSPs with dynamic content.

  The publishing facility can place HTML files, JSPs, and servlet code into appropriate directories for running in the Application Server or for testing with VisualAge for Java.

  WebSphere Studio also provides direct interaction with VisualAge for Java to store and retrieve Java and class files into and from the VisualAge for Java repository.
- Web applications can access enterprise resources, such as DB2, CICS, MQSeries, IMS, SAP, and others, through connectors.

2 Product overview

In this chapter we give a brief overview of the products that are used throughout this book.

We will start with a short section on how the different products can be used in different areas, and then follow it up with an overview of each individual product. For installation instructions for the products, refer to Chapter 3, “Product installation” on page 17.

How the products work together

The diagram in Figure 6 provides a picture of how the different products that we will be using in this book can work together in a complete development environment. More detailed information on usage of the major products can be found in later chapters.

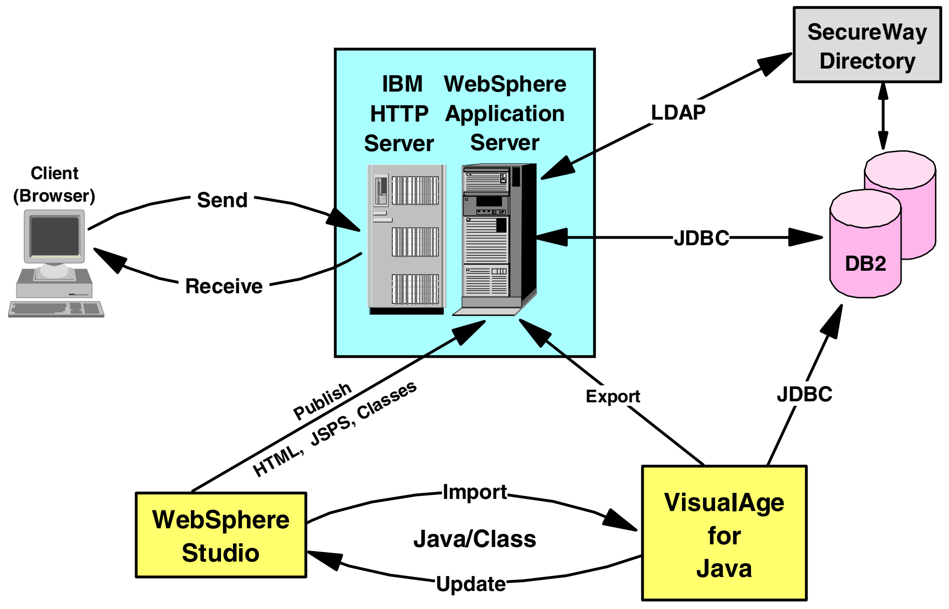


Figure 6. Products in our development environment

IBM HTTP Server

IBM HTTP Server powered by Apache is based on the Apache HTTP Server and runs on AIX, Solaris, Windows NT, and Linux.

IBM has enhanced the Apache-powered HTTP Server; for example, IBM added SSL for secure transactions.

For more information on IBM HTTP Server, see the product documentation and visit the Web site:

http://www.ibm.com/software/webservers/httpservers/

WebSphere Application Server

WebSphere Application Server (WAS) allows you to extend the functionality of a standard Web server. WAS enables Web transactions and interactions with a robust deployment environment for e-business applications. It provides a portable, Java-based Web application deployment platform focused on supporting and executing servlets, JavaBeans, and JavaServer Pages (JSP) files.

In particular, the **Standard Edition**, for Web site builders, provides:

- ❑Support for JavaServer Pages, including:
  - Support for specifications 0.91 and 1.0
  - Extended tagging support for queries and connection management
  - An Extended Markup Language (XML)-compliant DTD for JSPs
- ❑Support for the Java Servlet API 2.1 specification, including automatic user session and user state management
- ❑High speed pooled database access using JDBC for DB2 Universal Database, Oracle and Microsoft SQLServer
- ❑XML server tools, including a parser and data transformation tools
- ❑A Web site analysis tool for developing traffic measurements to help improve the performance and effectiveness of your Web sites
- ❑Machine translation for dynamic language translation of Web page content
- ❑Tivoli-ready modules
- ❑Additional integration with IBM VisualAge for Java to help reduce development time by allowing developers to remotely test and debug Web-based applications

The **Advanced Edition**, for Web application programmers, provides all the features of the Standard Edition, plus:

- ❑Full support for the Enterprise JavaBeans (EJB) 1.0 specification
- ❑Deployment support for EJBs, Java servlets, and JSPs with performance and scale improvements, including:
  - Applet-level partitioning
  - Load balancing
- ❑Enhanced support for distributed transactions and transaction processing
- ❑Improved management and security controls, including:
  - User and group level setup
  - Method level policy and control
- ❑CORBA support, providing both bean-managed and container-managed persistence

The **Enterprise Edition**, for Web enterprise architects, includes all the features of the Advanced Edition, plus:

- ❑Full distributed object and business process integration capabilities
- ❑IBM's world-class transactional application environment integration (from TXSeries)
- ❑Full support for the Enterprise JavaBeans (EJB) 1.0 specification
- ❑Complete object distribution and persistence (from Component Broker)
- ❑Support for MQSeries
- ❑Complete component backup and restore support
- ❑XML-based team development functions
- ❑Integrated Encina application development kit

For more information on WebSphere Application Server, see the product documentation and visit the Web site:

`http://www.ibm.com/software/webservers/appserv/`

WebSphere Studio

WebSphere Studio is an integrated suite of tools and wizards for building, organizing, and deploying Web applications in a team environment. Studio combines graphical development wizards with tools for Web site design and limited Java development, with integrated features including:

- ❑A workbench environment that lets Web development teams organize and manage Web development projects. This environment can be extended with source control management (SCM) tools.
- ❑A visual page designer for JSPs, HTML, and DHTML.
- ❑A remote debugger for easy remote debug of server-side scripts and logic, including JSP components, servlets, JavaBean components, and more. The remote debugger requires the Standard or Advanced edition of IBM WebSphere Application Server.
- ❑Wizards to help developers generate JSPs, JavaBeans, SQL statements, and servlets.
- ❑Integration between IBM VisualAge for Java Professional Edition, V3.0 and Studio.
- ❑An applet designer based on the NetObjects BeanBuilder technology.
- ❑NetObjects ScriptBuilder for script editing of Extensible Markup Language (XML) and Wireless Markup Language (WML).

- ❑A Web art designer for creating masthead images, buttons and other graphics.
- ❑An animated GIF designer that makes it easier to create animated GIFs.

For more information on WebSphere Studio, see the product documentation and visit the Web site:

`http://www.ibm.com/software/webservers/studio/`

VisualAge for Java

VisualAge for Java is IBM's Java development environment. It is an integrated, visual development environment with powerful support for JavaBeans, client/server development, visual programming and enterprise connectivity.

These are three VisualAge for Java editions: Entry, Professional, and Enterprise.

- ❑VisualAge for Java Entry Edition is a free version with a 750 class limit. This makes it ideal for small projects or evaluation purposes.
- ❑VisualAge for Java Professional Edition removes the 750 class limit from the Entry edition.
- ❑VisualAge for Java Enterprise Edition adds enterprise access builders and a team programming environment to the Professional Edition.

Common to all editions are:

- ❑Incremental compilation
- ❑Visual Composition Editor—for visual programming
- ❑Integrated Development Environment, including:
  - Debugger
  - Browsers—Project, Package, and Class
  - Source code editor
- ❑Repository-based environment for code-management
- ❑Advanced coding tools, including automatic formatting, automatic code completion, and fix-on-save
- ❑Data Access Beans for simplified access to relational databases

For more information on VisualAge for Java, see the product documentation and visit the Web site:

`http://www.ibm.com/software/ad/vajava/`

Distributed Debugger

The IBM Distributed Debugger is a client/server application that enables you to detect and diagnose errors in your programs. This client/server design makes it possible to debug programs running on systems accessible through a network connection as well as debug programs running on your workstation. The Distributed Debugger comes with VisualAge for Java.

For more information on the Distributed Debugger, see the product documentation.

DB2 Universal Database (UDB)

DB2 UDB is a relational database management system. It is fully scalable, being able to grow from single processors through symmetric multiprocessors up to massively parallel clusters. It has full multimedia capabilities, being able to support image, audio, video, text, and other advanced object support. It is also very Web-enabled, including built-in Java support.

For more information on DB2 UDB, see the product documentation and visit the Web site:

`http://www.ibm.com/software/data/db2/udb/`

SecureWay Directory

SecureWay Directory is a Lightweight Directory Access Protocol (LDAP) based directory server that provides a common and simple method for centrally storing, locating and managing directory information on an enterprise network across multiple platforms. It also provides security services allowing you to define user access rights for the information stored in a directory.

For more information on SecureWay Directory, see the product documentation and visit the Web site:

`http://www.ibm.com/software/network/directory/`

3 Product installation

In this chapter we describe the installation process for the various products that are used throughout this book.

We will discuss the environment that we are using in developing this book and the applications that are part of it. We will also step through any other setup instructions that are required prior to installing the products.

Next, we will guide you through an example installation of each product, and then show you how you can test that each product has been successfully installed.

Starting environment

The examples in this book were developed in the following environment:

- ❑PCs with Pentium II 450 MHZ processors and 512 MB RAM.
- ❑MS Windows NT 4.0 with Service Pack 4.
- ❑A combination of Netscape Communicator 4.61 and Microsoft Internet Explorer 5.0 were used in testing the various components.

Due to various interdependencies between products, the order in which you install the following software is important. Although other variations may also work, we used the following to ensure a successful setup.

All of our work for this book was carried out on a *d* drive. However, to use a standard naming convention, throughout this book we refer to the *x* drive when listing paths. You should substitute your own drive letter in as appropriate.

Creating a dedicated user ID

Many of the following products require a user ID under which to run certain administrative tasks (and some will not allow you to use the *Administrator* ID in Windows NT for that task).

Therefore, it is a good idea to create a dedicated user ID for all of these products to use. This user will need to have full Administrator authority to work, and so you should make it a member of the Administrator group for your machine.

So that WebSphere Application Server can use this user ID for all its needs, you should also make sure that it has the rights to *Log on as a service* and to *Act as part of the operating system*. Also, WebSphere Application Server will not let you use an account whose name matches the name of your machine or Windows Domain. If you need help in doing this, then refer to your Windows Help documentation.

For the purposes of this book, we created a user ID called *itso* with the password *itso*. It is not necessary that you create such a user ID, but make sure that the user ID used for WebSphere has full administrative authorities.

Java Development Kit

We installed IBM's JDK 1.1.7 for Windows on to our machines. The refresh level that we have used for this book is *ibm-jdk-n117p-win32-x86* and is the latest available at the time of writing. IBM Java developer kits for a variety of platforms can be found at `http://www.ibm.com/java/jdk/download`.

To test your installation, open up a command prompt and enter:

```
x:\>java -fullversion
```

It should return something similar to the following (though build numbers may vary):

```
java full version “JDK 1.1.7 IBM build xxxxx-yyyymmdd (JIT enabled: ibmjitc)”
```

IBM HTTP Server

IBM HTTP Server Version 1.3.6.2 was installed to run the sample applications in this book.

Installing the product

IBM HTTP Server has a fairly straightforward installation program. When running it, select a *typical* installation, and when prompted, enter the user ID and password you created in “Creating a dedicated user ID” on page 18.

Testing the install

After installation, you can carry out a quick check to ensure that the server is up and running. Open up a Web browser, and enter the URL, `http://localhost`. You should see the HTTP Server welcome page, as displayed in Figure 7.

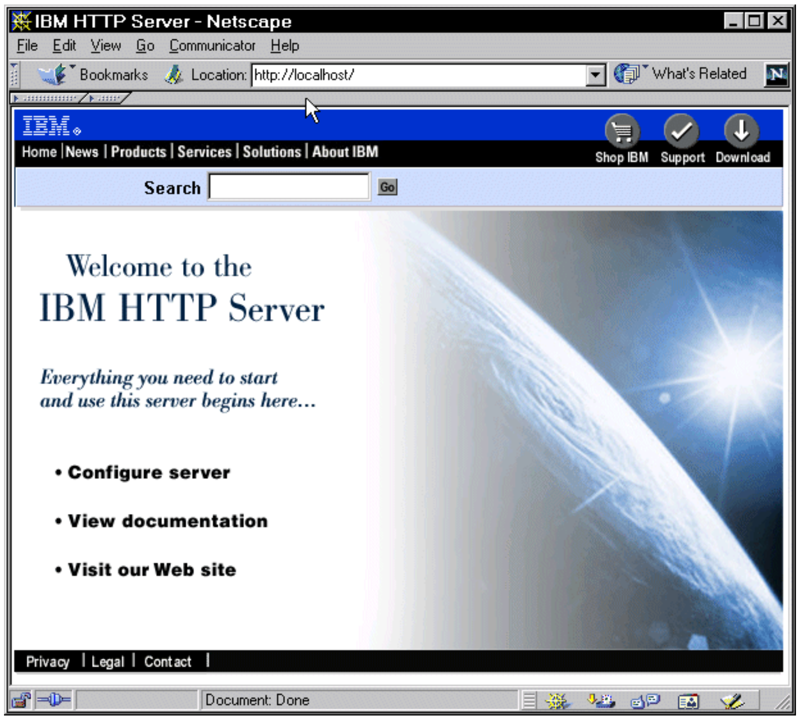

IBM HTTP Server - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/ What's Related
IBM
Home | News | Products | Services | Solutions | About IBM
Shop IBM Support Download
Search Go
Welcome to the
IBM HTTP Server
Everything you need to start and use this server begins here...
• Configure server
• View documentation
• Visit our Web site
Privacy | Legal | Contact |
Document: Done

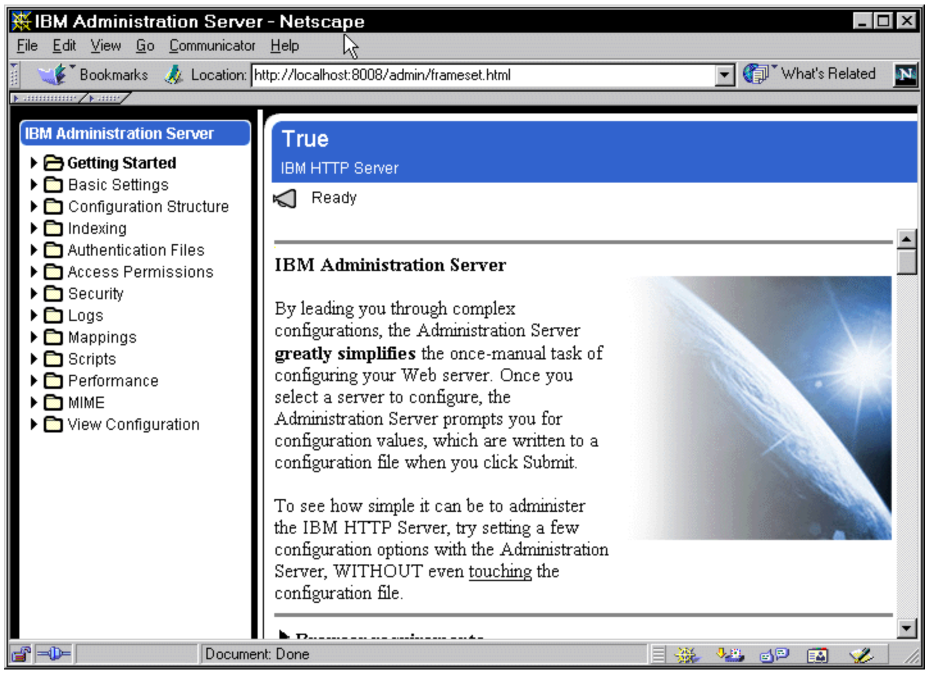
Figure 7. IBM HTTP Server welcome page

At this point, you can set up the administration user ID and password for IBM HTTP Server to be something other than the default. The simplest way to do this is at a command prompt. Change to the IBM HTTP Server directory and enter:

```
htpasswd -c conf/admin.passwd userid
```

In this command, *userid* should be replaced with the administration user ID you want to use. You will then be prompted to enter the password twice, and the system will set these values. You can test this by entering the URL, `http://localhost:8008`. The system will prompt you for the administration user ID and password, and if you enter these correctly, you should see a page similar to that in Figure 8.

Note that IBM HTTP Administration is a service separate from IBM HTTP Server. Both services can be started and stopped from the *Services* icon in the Control Panel.



IBM Administration Server

By leading you through complex configurations, the Administration Server **greatly simplifies** the once-manual task of configuring your Web server. Once you select a server to configure, the Administration Server prompts you for configuration values, which are written to a configuration file when you click Submit.

To see how simple it can be to administer the IBM HTTP Server, try setting a few configuration options with the Administration Server, WITHOUT even touching the configuration file.

Figure 8. Administration interface to IBM HTTP Server

DB2 Universal Database

We installed DB2 Universal Database for Windows NT Version 6.1 Workgroup edition and added the latest Fixpack.

Installing the product

To keep things simple, we used the *typical* installation option to install this product.

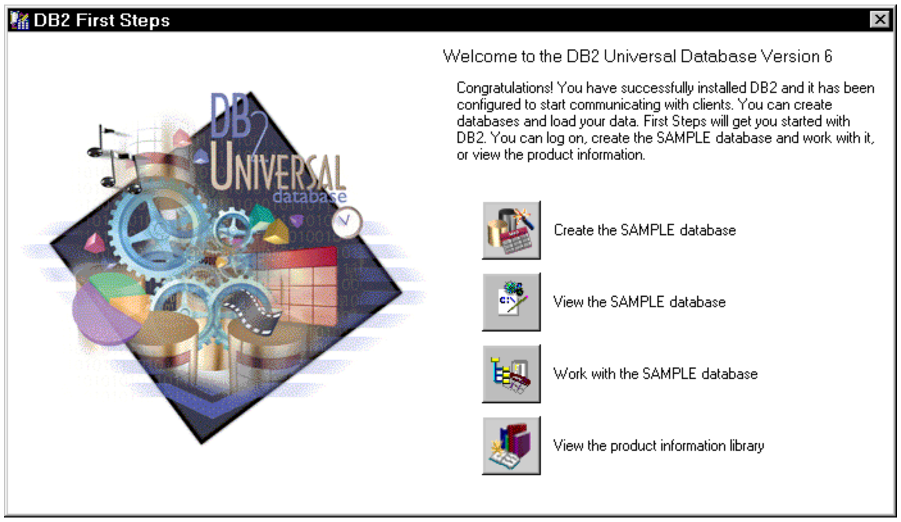
The DB2 administration server needs a user ID that has *Administrator* rights to your machine to be able to run, and you should use the one created in “Creating a dedicated user ID” on page 18.

The install will prompt to you reboot your machine. As you want to install the latest Fixpack as well, it will save time if you choose to reboot later. At the time of writing, the latest Fixpack for DB2 Version 6.1 is Fixpack 2.[1]

[1] The Fixpack level had to be changed on machines where we ran SecureWay Directory (see “Incompatibilities with DB2 UDB” on page 35).

Run the install for the Fixpack, and this time, when prompted, you should choose to reboot the machine.

When your machine restarts, you should be greeted with the DB2 First Steps window, as shown in Figure 9.


DB2 First Steps
DB2 Universal database
Create the SAMPLE database
View the SAMPLE database
Work with the SAMPLE database
View the product information library

Welcome to the DB2 Universal Database Version 6

Congratulations! You have successfully installed DB2 and it has been configured to start communicating with clients. You can create databases and load your data. First Steps will get you started with DB2. You can log on, create the SAMPLE database and work with it, or view the product information.

Figure 9. DB2 First Steps window

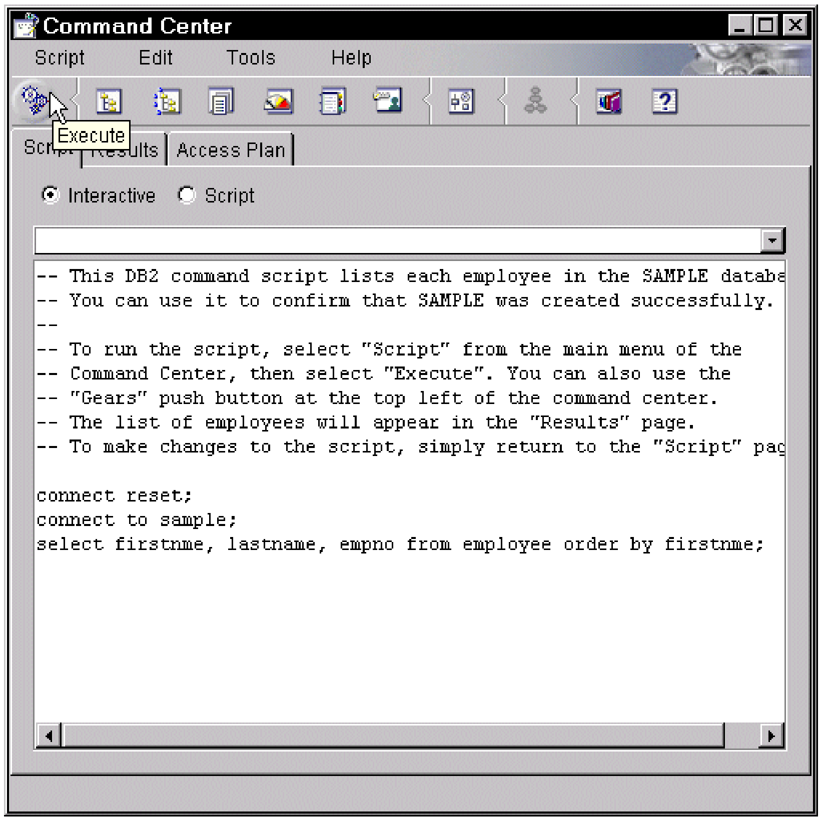
You should select *Create the SAMPLE database*. This will set up a sample database for you in your DB2 installation, which can allow you to test your access to DB2.

Testing the installation

To test your installation you can select *View the SAMPLE database* from the First Steps window (if you have closed this window, you can re-open it from the *DB2 for Windows NT* menu on the Start Bar).

You will be prompted for your DB2 login, and then the Command Center will open up. This has a script pre-loaded that will access the sample database. You can execute the script by clicking on the *gears* icon, as shown in Figure 10.

If this displays the list of employees stored in the sample database, then your install has been successful.



```
-- This DB2 command script lists each employee in the SAMPLE databa
-- You can use it to confirm that SAMPLE was created successfully.
--
-- To run the script, select "Script" from the main menu of the
-- Command Center, then select "Execute". You can also use the
-- "Gears" push button at the top left of the command center.
-- The list of employees will appear in the "Results" page.
-- To make changes to the script, simply return to the "Script" pag

connect reset;
connect to sample;
select firstnme, lastname, empno from employee order by firstnme;
```

Figure 10. Script for viewing the sample database

The tables in the sample database are prefixed by the user ID that creates the database. This makes it hard to write examples that run on every system. Our examples use table names such as ITSO.DEPARTMENT. If you create the database with a different user ID, you can assign aliases to make the examples work. Run these commands in a DB2 Command Window:

```
db2 connect to sample
db2 create alias itso.department for department
db2 create alias itso.employee for employee
db2 create alias itso.emp_photo for emp_photo
db2 connect reset
```

VisualAge for Java

VisualAge for Java Version 3.02 Enterprise Edition was installed on our machines, but we could have used the Professional Edition, as this also provided the functionality required for this book.

Installing the product

When running the setup, select a *full* installation and select *Local* for the location of your repository, as we are not going to be setting up a team environment.

When all the files are installed, you will then be prompted to reboot. Do so and then start up VisualAge for Java to complete the basic installation. During start-up, you will be asked to select the workspace owner and enter their network name. Select *Administrator* and enter your normal NT logon as the network name, then click OK.

VisualAge will then finish adding files to your workspace. When it has finished doing this, you will be prompted with the Welcome to VisualAge window. Select *Go to the Workbench* and click OK (see Figure 11).

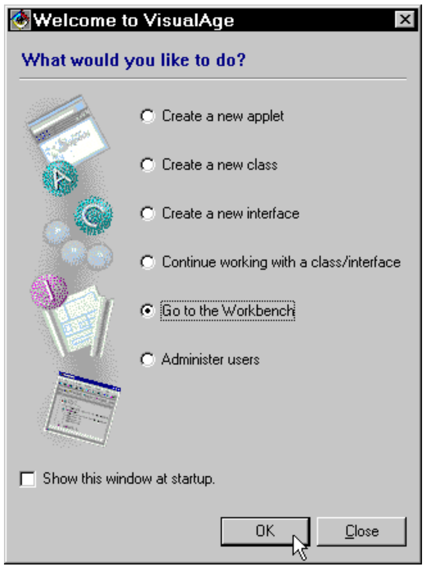


Figure 11. Welcome to VisualAge

Adding features

To enable you to create and run some of the applications within this book, you have to add some extra features to your VisualAge for Java installation. These will add some extra tools and code packages to your workspace.

To add the features, select *File -> Quick Start* and wait for the Quick Start window. Select *Features* and select *Add Feature*, as in Figure 12.

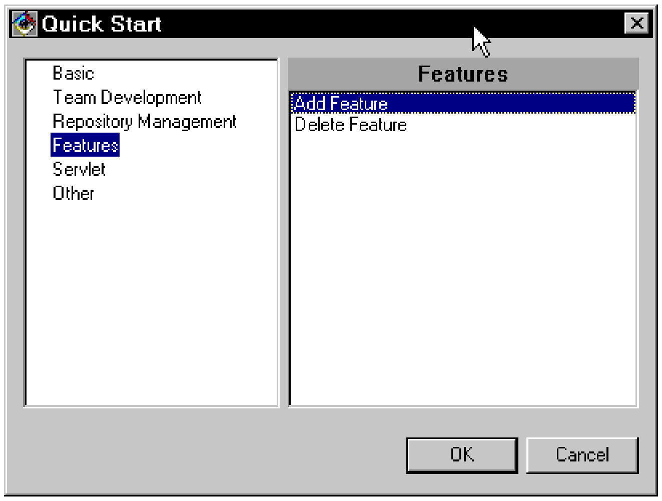


Figure 12. Adding a feature in VisualAge for Java

Select the following features and then click on *OK*:

❑IBM JSP Execution Monitor 1.1

❑IBM WebSphere Test Environment 3.02

When these features are loaded, your VisualAge for Java installation is ready for our examples.

Testing the installation

For a quick visual test that the installation has been successful, go to the menu option *Workspace -> Tools*. In the list of tools you should find the two options *JSP Execution Monitor* and *Launch WebSphere Test Environment*.

Existing errors

After installing VisualAge for Java, you may notice that there are some errors in the pre-loaded classes, specifically in the package `com.sun.java.swing.plaf.mac`. These errors will not effect any of the work required for this book, and you can ignore them.

Distributed Debugger

We also chose to install IBM Distributed Debugger Version 8.4 which comes with VisualAge for Java 3.02.

We chose a *full* install, which gave us the Object Level Trace facility as well, and can be used within WebSphere Application Server as well as via VisualAge for Java.

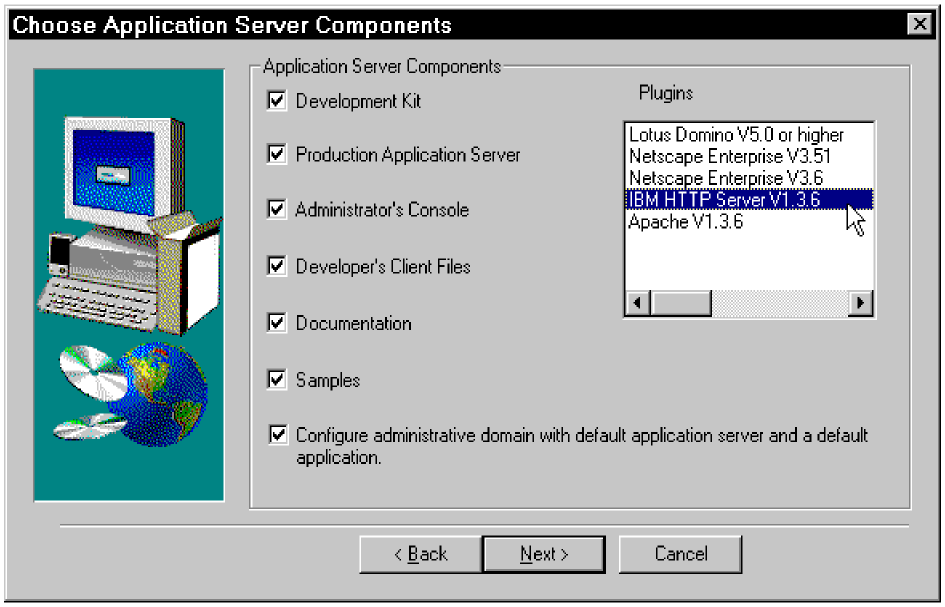
WebSphere Application Server

We installed WebSphere Application Server Version 3.02 Standard and Advanced Edition for the purposes of this book. Standard Edition would be enough as the additional features in the Advanced and Enterprise Editions were not required.

Installing the product

When you are running the install, choose the *Custom* installation and follow these steps:

❑You will be presented with the screen found in Figure 13. Keep all the components selected on the left, and make sure you select IBM HTTP Server V1.3.6 as the Plugin server. Then click on *Next*.



Choose Application Server Components

Application Server Components

☑ Development Kit

☑ Production Application Server

☑ Administrator's Console

☑ Developer's Client Files

☑ Documentation

☑ Samples

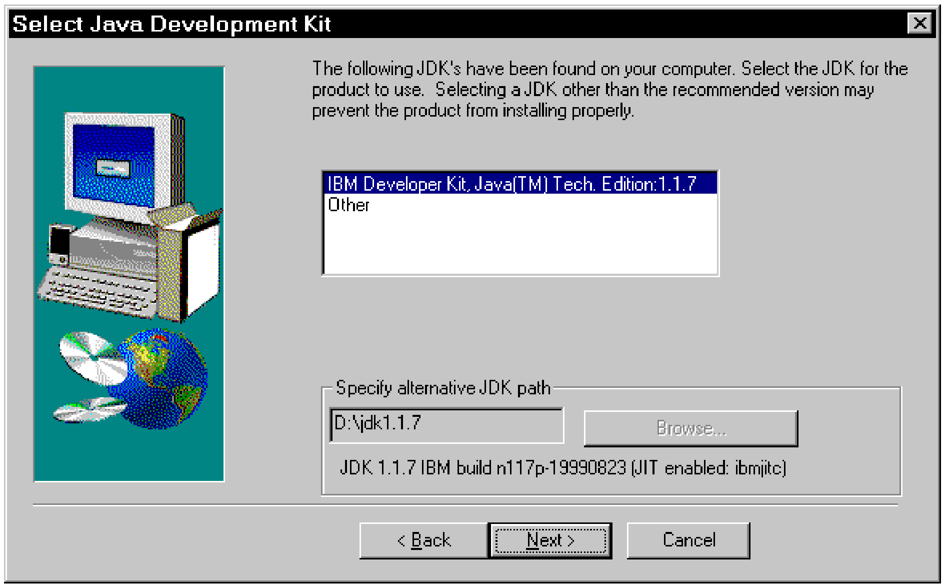
☑ Configure administrative domain with default application server and a default application.

Plugins

- Lotus Domino V5.0 or higher
- Netscape Enterprise V3.51
- Netscape Enterprise V3.6
- IBM HTTP Server V1.3.6
- Apache V1.3.6

Figure 13. Custom installation for WebSphere Application Server

❑You should then be presented with a screen prompting you to select a JDK for WebSphere to use (see Figure 14). The IBM Developer Kit, Java Tech Edition, that you installed earlier should appear in the list; you should select it and click on *Next*. If this item does not appear, then it is a good idea to come out of this installation and re-install the IBM JDK to make sure that WebSphere picks it up properly.


Select Java Development Kit

The following JDK's have been found on your computer. Select the JDK for the product to use. Selecting a JDK other than the recommended version may prevent the product from installing properly.

IBM Developer Kit, Java(TM) Tech. Edition:1.1.7
Other

Specify alternative JDK path

D:\jdk1.1.7

JDK 1.1.7 IBM build n117p-19990823 (JIT enabled: ibmjitc)

Figure 14. Selecting a JDK within the WebSphere install

❑You will then be presented with the Security/Database options screen as shown in Figure 15. For the Security section, you can enter the user ID and password that you created in “Creating a dedicated user ID” on page 18. Next, change the *Database Type* to DB2; this should enable you to enter a user ID and password for the database. Again, you can use the same user ID that you created in “Creating a dedicated user ID” on page 18.[2] This will be the user ID under which WebSphere creates the WAS database.[3] You have to enter a user ID that has full administrative authorities. Click on *Next*.

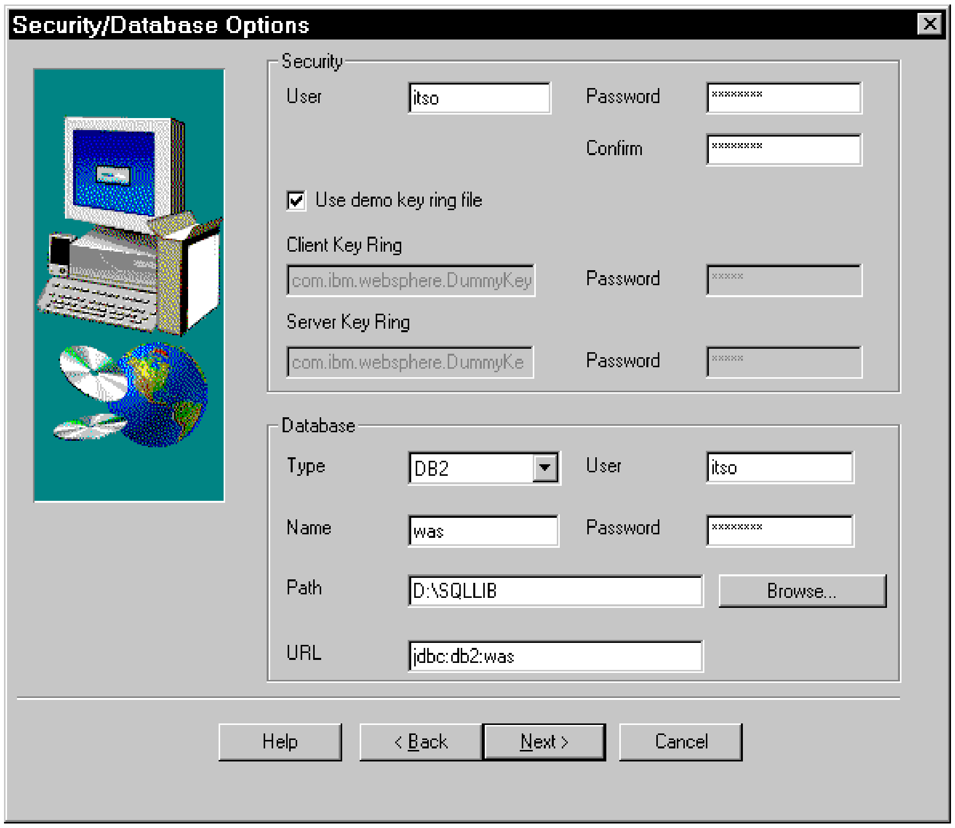
❑Click *Next* on the following confirmation, and installation of the files will start.

❑Towards the end of the install, it will ask you for the location of the IBM HTTP Server configuration file. This should have a path similar to:

```
d:/IBM HTTP Server/conf/httpd.conf
```

If you have installed the HTTP Server, the installation process should automatically find this file for you, and you can click *OK* to finish the install process.

[2] By using the same IDs for both sections, you can help avoid problems later on when starting WebSphere AdminServer as a service.
[3] The WAS database is used by WebSphere to store the configuration information for your server components.



Security

User: itso

Password: *********

Confirm: *********

☑ Use demo key ring file

Client Key Ring: com.ibm.websphere.DummyKey

Password: *****

Server Key Ring: com.ibm.websphere.DummyKe

Password: *****

Database

Type: DB2

User: itso

Name: was

Password: *********

Path: D:\SQLLIB

URL: jdbc:db2:was

Figure 15. Setting up Security and Database options for WebSphere

❑After rebooting the machine as prompted, the install script creates the WAS database in your DB2 system. This database is used by WebSphere Application Server to manage its configuration information.

You can create the WAS database manually if the automatic process fails, for example, because DB2 was not started. Use the command file:

```
d:\WebSphere\AppServer\bin\createdb2.bat
```

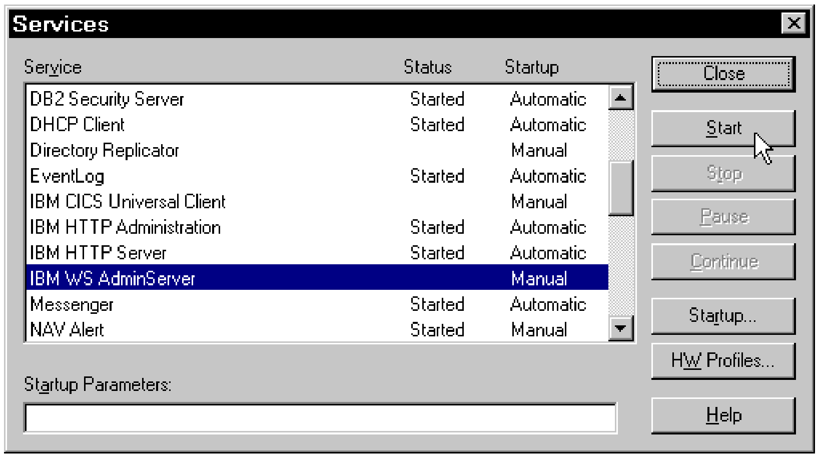
You have now completed the installation of WebSphere Application Server.

Testing the installation

To test the installation, you have to start up the WS AdminServer that enables you to administer WAS.

Starting the WS AdminServer service

You start the administration server through the Control Panel - Services window as shown in Figure 16. Once this server is started, you will be able to administer your servers, including starting the default server.



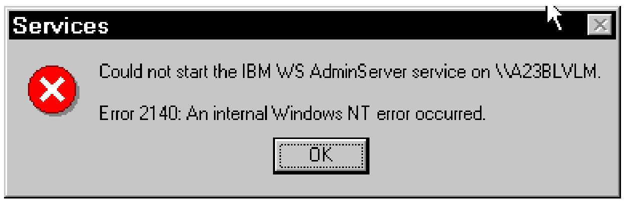
Service	Status	Startup
DB2 Security Server	Started	Automatic
DHCP Client	Started	Automatic
Directory Replicator		Manual
EventLog	Started	Automatic
IBM CICS Universal Client		Manual
IBM HTTP Administration	Started	Automatic
IBM HTTP Server	Started	Automatic
IBM WS AdminServer		Manual
Messenger	Started	Automatic
NAV Alert	Started	Manual

Figure 16. Starting WebSphere Application Server as a service

It is a good idea to set this service up to start automatically in the future. You can do this by clicking on *Startup* while the WS AdminServer is selected and, in the following dialog, changing the Startup Type to *Automatic*.

Errors when starting the WS AdminServer

A problem that is sometimes encountered when starting the WS AdminServer is that you will be given the error window shown in Figure 17.



Could not start the IBM WS AdminServer service on \\A23BLVLM.

Error 2140: An internal Windows NT error occurred.

Figure 17. Error when starting the WS AdminServer

This can be caused by a number of problems, but a common one that we found results from having used a different user ID and password combination for the database when installing WebSphere. If this is the case, then you can fix the error by editing:

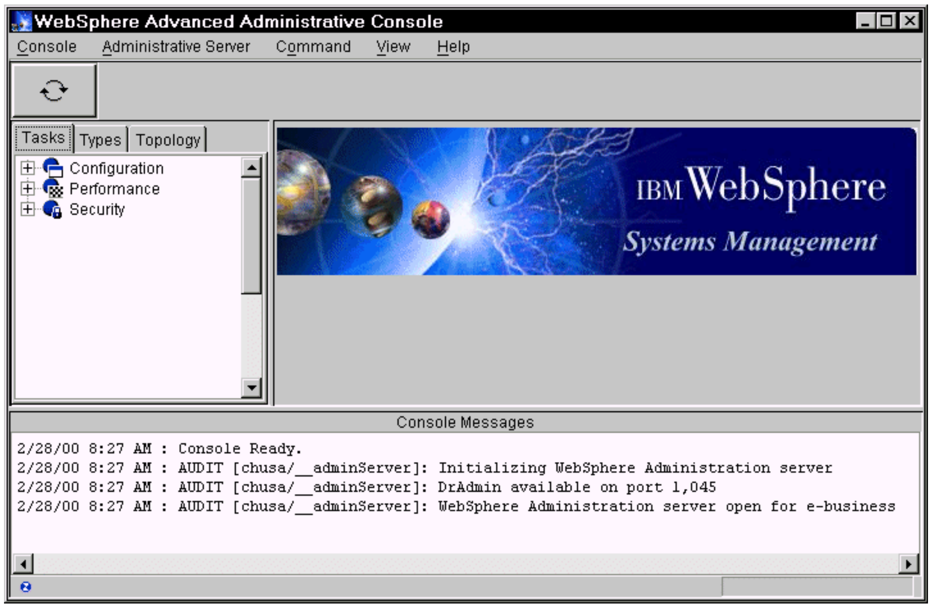
```
d:\WebSphere\AppServer\bin\admin.config
```

Change the *dbUser* and *dbPassword* entries to be the same ID that you have used elsewhere.

Starting the Administrative Console

Once the WS AdminServer is successfully started, you can use the administration console to configure different components of WebSphere. This program can be started from the Start menu by selecting the *Administrator's Console* in the *IBM WebSphere -> Application Server 3.0* folder.

Figure 18 shows the Administrative Console window, and when the spinning icon at the bottom comes to a stop, the program is fully loaded and ready.



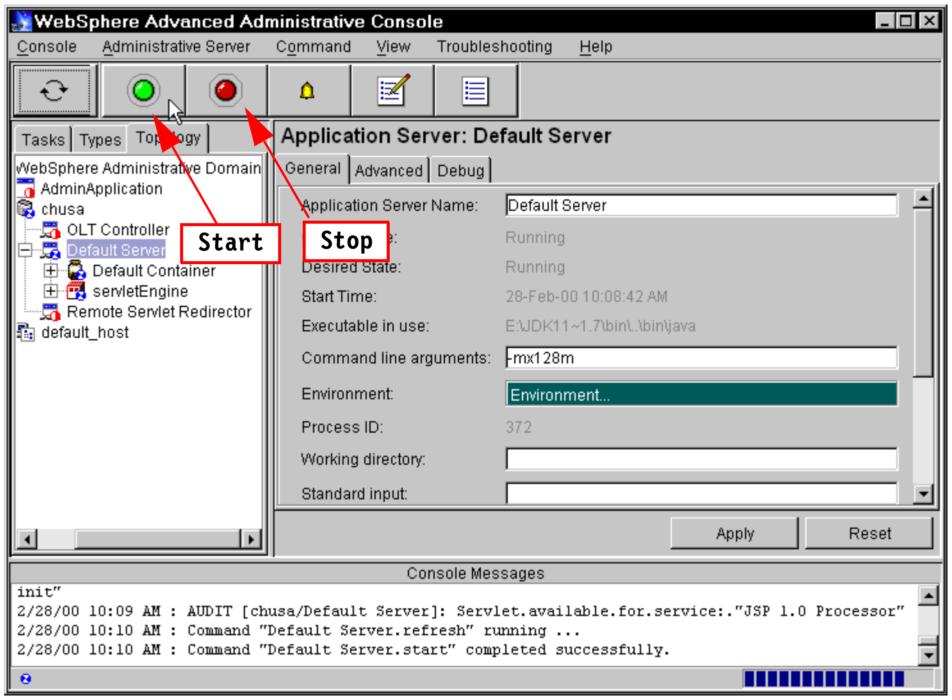
```
2/28/00 8:27 AM : Console Ready.
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: Initializing WebSphere Administration server
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: DrAdmin available on port 1,045
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: WebSphere Administration server open for e-business
```

Figure 18. The WebSphere Administrative Console

From here we can manage our servers completely. For now, though, we just want to start the Default Server to test that everything has installed correctly. For more information on how to use the administration interface, refer to Chapter 6, “WebSphere Application Server” on page 123.

Starting the Default Server

If you select the *Topology* tab, and then click on the + sign next to WebSphere Admin Domain, you should see the name of your machine. Expanding your machine name, you will find the Default Server as shown in Figure 19. When you select the Default Server, you can then click on the button with a green light symbol to start it. After a while, a dialog should come up telling you the start command was successful.



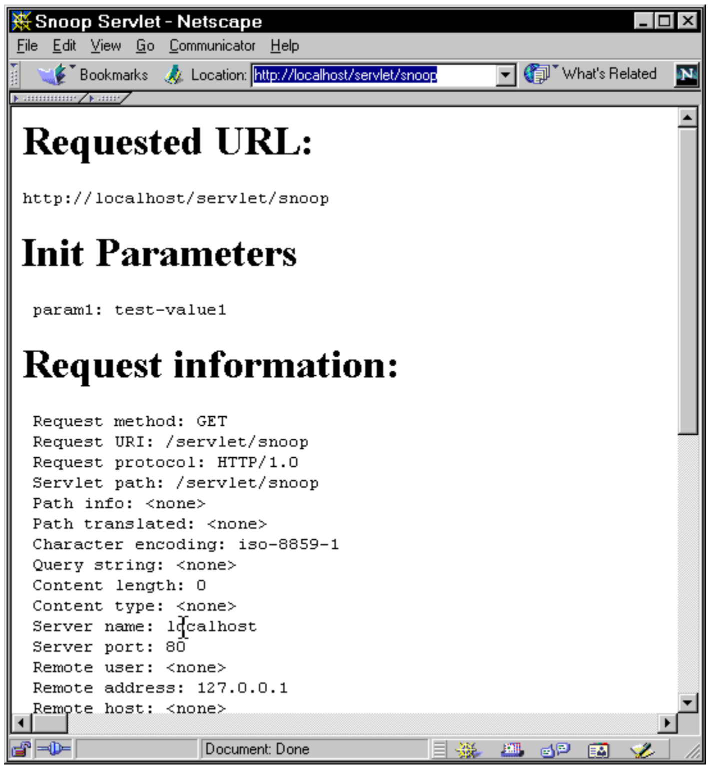
Application Server Name: Default Server
Current State: Running
Desired State: Running
Start Time: 28-Feb-00 10:08:42 AM
Executable in use: E:\JDK11~1.7\bin\..\bin\java
Command line arguments: -mx128m
Environment: Environment...
Process ID: 372
Working directory:
Standard input:

```
init"
2/28/00 10:09 AM : AUDIT [chusa/Default Server]: Servlet.available.for.service:."JSP 1.0 Processor"
2/28/00 10:10 AM : Command "Default Server.refresh" running ...
2/28/00 10:10 AM : Command "Default Server.start" completed successfully.
```

Figure 19. Starting the Default Server

Running a test servlet

Now that you have the Default Server running, you can quickly test it by running the Snoop servlet that is part of the standard install for WAS. Open up a browser and enter `http://localhost/servlet/snoop`. If the install has been completely successful, you should see a page similar to Figure 20.


Snoop Servlet - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/servlet/snoop
What's Related
Document: Done

```
Requested URL:

http://localhost/servlet/snoop

Init Parameters

 param1: test-value1

Request information:

 Request method: GET
 Request URI: /servlet/snoop
 Request protocol: HTTP/1.0
 Servlet path: /servlet/snoop
 Path info: <none>
 Path translated: <none>
 Character encoding: iso-8859-1
 Query string: <none>
 Content length: 0
 Content type: <none>
 Server name: localhost
 Server port: 80
 Remote user: <none>
 Remote address: 127.0.0.1
 Remote host: <none>
```

Figure 20. Output from the Snoop servlet

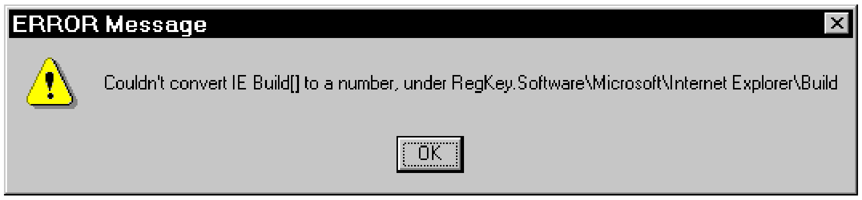
WebSphere Studio

We installed WebSphere Studio Version 3.0 and then applied Fixpack 2 to bring it up to Version 3.0.2.

Installing the product

While installing Studio, we were prompted to see if we wanted to install the Applet Designer. Although it would not have done any harm, we elected not to do so, as it is not required for the topics covered in this book.

If you do not have a valid installation of Internet Explorer on your machine prior to installing Studio, you may encounter the errors shown in Figure 21 and Figure 22.

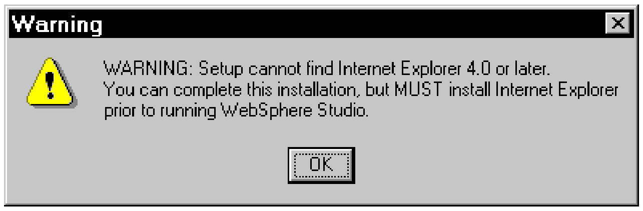


ERROR Message

Couldn't convert IE Build[] to a number, under RegKey.Software\Microsoft\Internet Explorer\Build

Figure 21. Error while installing WebSphere Studio

You can click OK on these screens and continue as, in contradiction to the message in Figure 22, it is not essential to be able to use Studio. But it will prevent you from directly previewing your pages in the Page Designer tool (see "Editing project resources" on page 237).



Warning

WARNING: Setup cannot find Internet Explorer 4.0 or later.
You can complete this installation, but MUST install Internet Explorer prior to running WebSphere Studio.

Figure 22. Warning information dialog concerning missing IE installation

Once completed, you have to reboot the machine and then run the setup program provided with Fixpack 2. After the install you have to reboot again and after that you have completed the installation of Studio.

Testing the installation

The easiest way to test the install of WebSphere Studio is to start it from the Start bar. Select *IBM WebSphere -> Studio 3.0 -> IBM WebSphere Studio v3.0*. The install was successful if the window shown in Figure 23 appears.

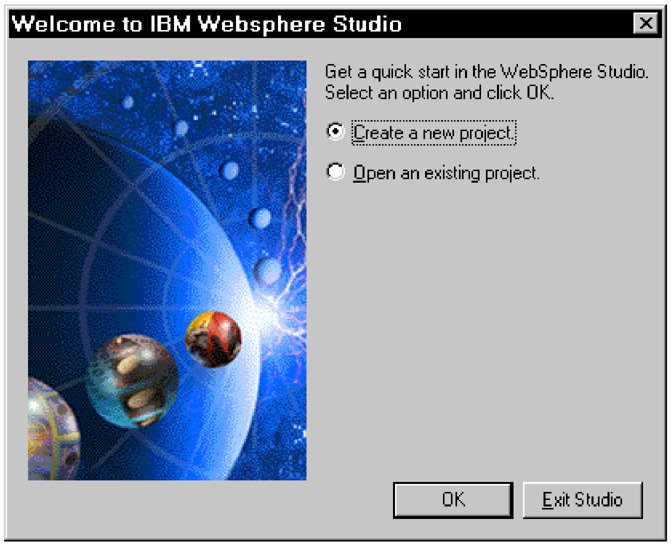

Welcome to IBM Websphere Studio
Get a quick start in the WebSphere Studio. Select an option and click OK.
Create a new project.
Open an existing project.
OK
Exit Studio

Figure 23. WebSphere Studio welcome window

SecureWay Directory

We installed SecureWay Directory Version 3.1.1, including the GS Kit, on our machines to help us with some of the applications in this book.

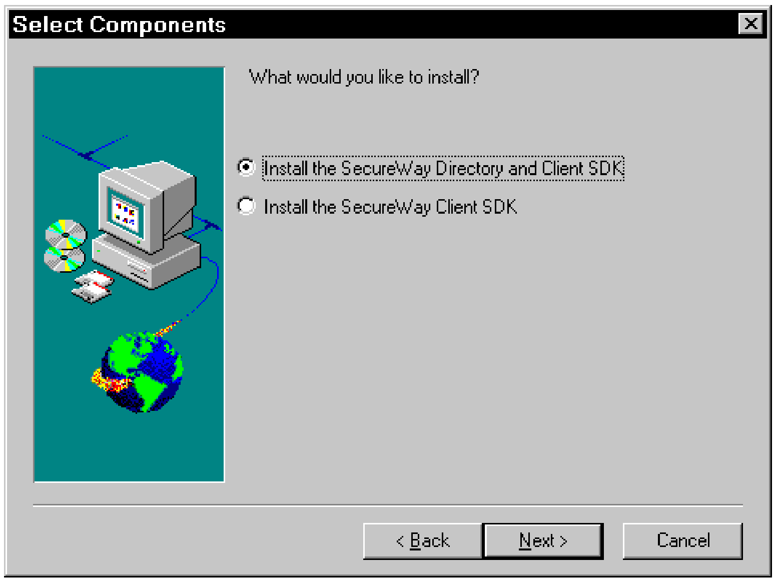
Incompatibilities with DB2 UDB

SecureWay Directory Version 3.1.1 does not function correctly with DB2 UDB Version 6.1 with Fixpack 2 applied. Therefore, on any machines that required the SecureWay Directory running, we had to downgrade our DB2 installation to Fixpack 1A. SecureWay Directory 3.1.1.5 has fixed this problem and works with Fixpack 2 of DB2 Version 6.1.

Installing the product

When installing SecureWay Directory, choose your appropriate installation language and then complete the following steps:

❑ Choose to install both SecureWay Directory and the Client SDK as in Figure 24.



Select Components

What would you like to install?

[✓ Install the SecureWay Directory and Client SDK]

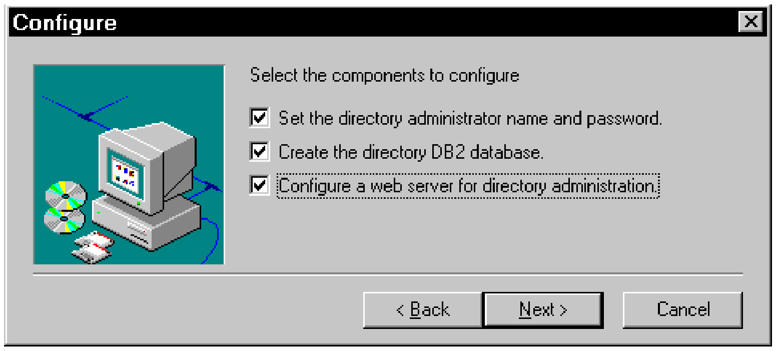
Install the SecureWay Client SDK

Figure 24. Selecting components to install

❑Choose the appropriate install directory (you can leave the default).

❑Choose the appropriate Program Folder (you can leave the default).

❑Select all three components to configure (Figure 25). You can actually do this configuration after the install is complete, but it is easier to do it here.



Configure

Select the components to configure

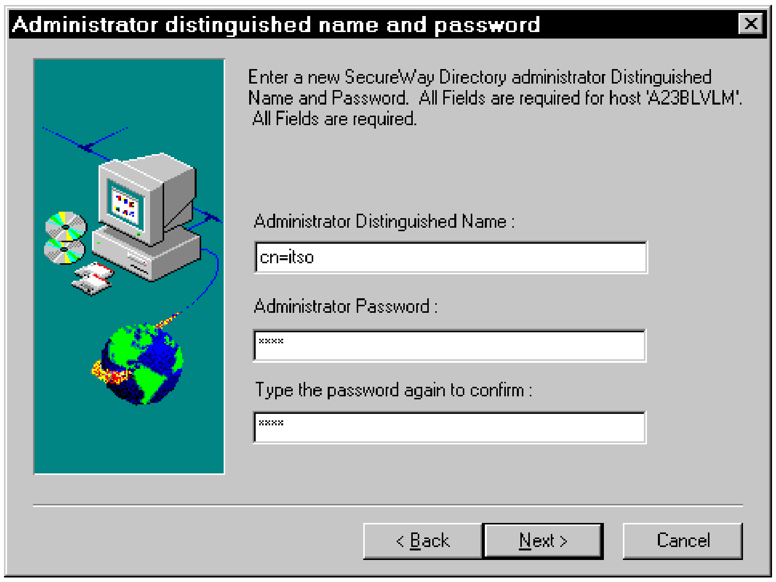
☑ Set the directory administrator name and password.

☑ Create the directory DB2 database.

☑ Configure a web server for directory administration.

Figure 25. Selecting the components to configure

❑Enter in a unique name and password for the SecureWay Directory administrator. To keep things simple, use the user ID and password that was created in “Creating a dedicated user ID” on page 18, but prefix the user ID with *cn*= (Figure 26).


Administrator distinguished name and password

Enter a new SecureWay Directory administrator Distinguished Name and Password. All Fields are required for host 'A23BLVLM'. All Fields are required.

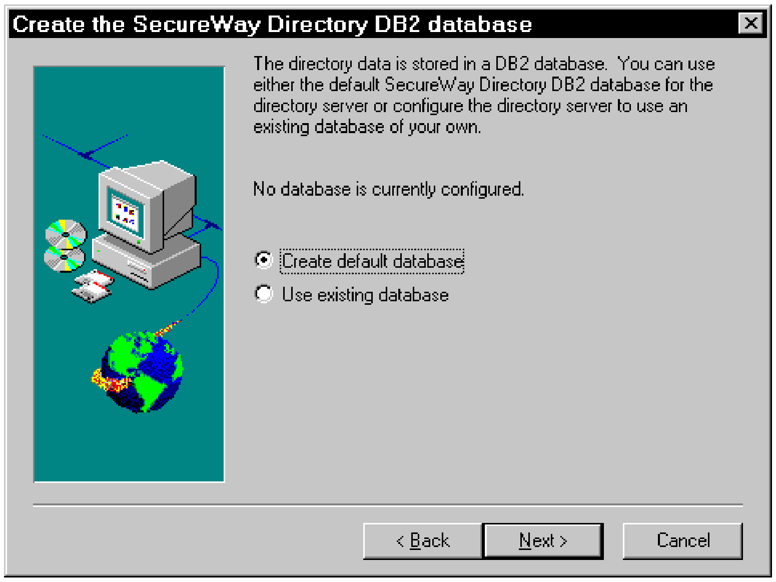
Administrator Distinguished Name : cn=itso

Administrator Password : ****

Type the password again to confirm : ****

Figure 26. Configuring the SecureWay Directory administrator

❑Choose to create the default SecureWay Directory database, as in Figure 27. When prompted, select to create the database using the local code page, and choose an appropriate drive for it to reside in.


Create the SecureWay Directory DB2 database

The directory data is stored in a DB2 database. You can use either the default SecureWay Directory DB2 database for the directory server or configure the directory server to use an existing database of your own.

No database is currently configured.

[✓ Create default database]
Use existing database

Figure 27. Creating the SecureWay Directory database

❑SecureWay then locates your Web server configuration file. If you have IBM HTTP Server installed, it should find the file without any problems, and you can click on *Next*.

❑Click on *Next* in the final confirmation dialog, and installation will start.

❑When the install has finished, reboot the machine as prompted.

❑After reboot, a database script runs, and when that is complete, you have installed SecureWay Directory.

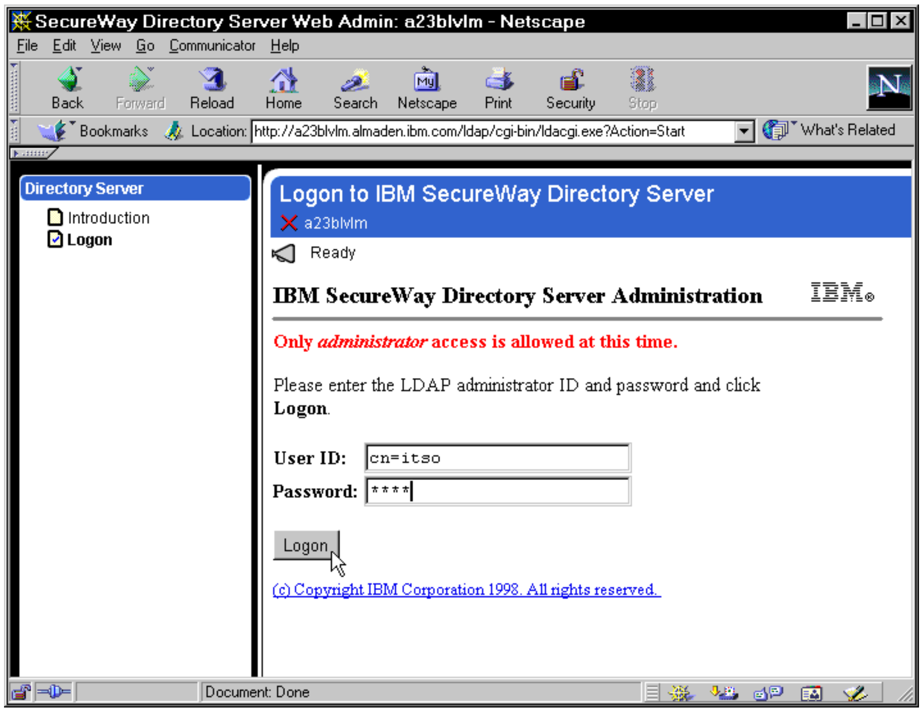
Configuring SecureWay Directory

After installation, you have to configure the server. First, open up a Web browser and enter the URL:

```
http://localhost/ldap
```

This should direct you to the administration login screen (see Figure 28).

Logging in


SecureWay Directory Server Web Admin: a23blvlm - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://a23blvlm.almaden.ibm.com/ldap/cgi-bin/ldacgi.exe?Action=Start
What's Related
Directory Server
Introduction
Logon
Logon to IBM SecureWay Directory Server
a23blvlm
Ready
Logon
Document: Done

IBM SecureWay Directory Server Administration

Only *administrator* access is allowed at this time.

Please enter the LDAP administrator ID and password and click **Logon**.

User ID: cn=itso
Password: ****

(c) Copyright IBM Corporation 1998. All rights reserved.

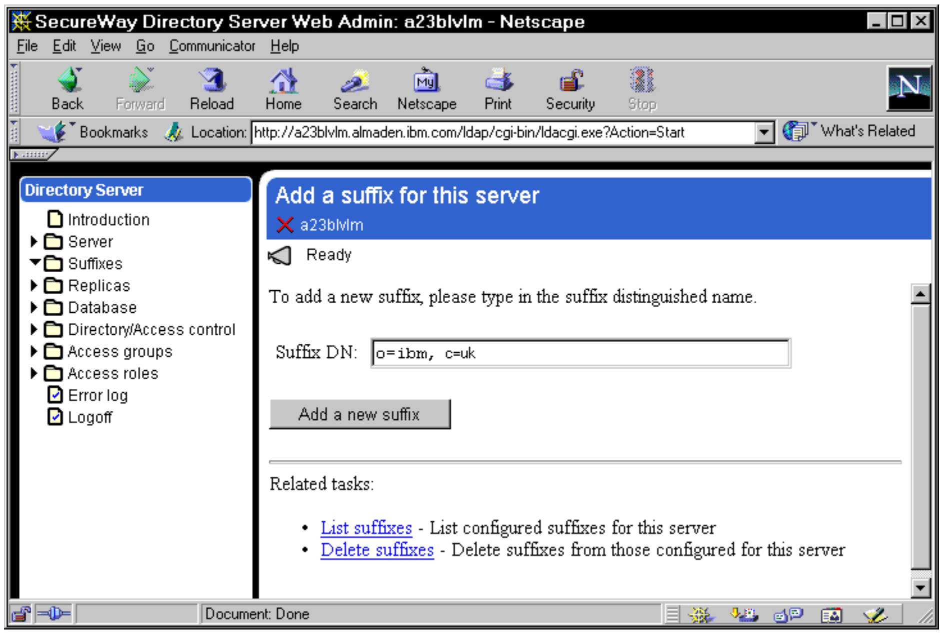
Figure 28. Logon to SecureWay Directory Server Administration

Enter in the user ID and password that you supplied during the install. (Remember to prefix the user ID with *cn=*). Then click on *Logon*. You should now be logged on to the server.

Adding suffixes

You now need to add a suffix to the configuration. This is required to set up the top-level entry for the directory hierarchy.

In the left hand panel, select *Add a suffix*, and you should see the *Add a suffix for this server* page in the right hand panel (see Figure 29). In the *Suffix DN* field, add *o=ibm, c=uk* and click on *Add a new suffix*.


SecureWay Directory Server Web Admin: a23blvlm - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://a23blvlm.almaden.ibm.com/ldap/cgi-bin/ldacgi.exe?Action=Start
What's Related
Directory Server
Introduction
Server
Suffixes
Replicas
Database
Directory/Access control
Access groups
Access roles
Error log
Logoff
Add a suffix for this server
a23blvlm
Ready
Add a new suffix
Related tasks:
Document: Done

To add a new suffix, please type in the suffix distinguished name.

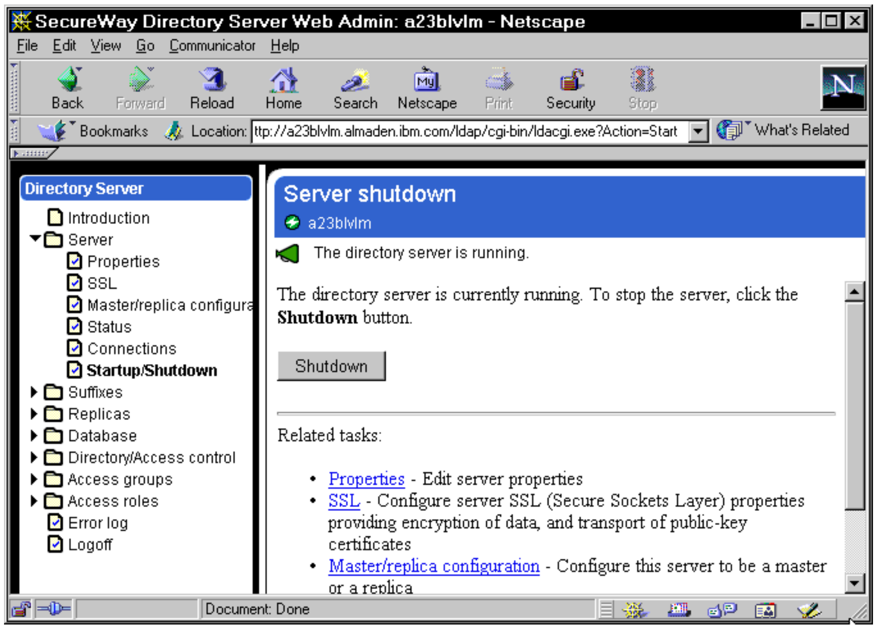
Suffix DN: o=ibm, c=uk

- List suffixes - List configured suffixes for this server
- Delete suffixes - Delete suffixes from those configured for this server

Figure 29. Adding a suffix

Starting the server

To allow your new suffixes to take effect, you have to restart the server. Navigate down *Server* in the left-hand pane, and select *Startup/Shutdown*. A message in the right-hand pane should tell you that *The directory server is currently stopped*. Click on the button labeled *Startup*, and you are informed that the server is being started. When the server is finally running, you should see a window as shown in Figure 30.


SecureWay Directory Server Web Admin: a23blvlm - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: ttp://a23blvlm.almaden.ibm.com/ldap/cgi-bin/ldacgi.exe?Action=Start What's Related
Directory Server
Introduction
Server
Properties
SSL
Master/replica configura
Status
Connections
Startup/Shutdown
Suffixes
Replicas
Database
Directory/Access control
Access groups
Access roles
Error log
Logoff
Shutdown
Document: Done

Server shutdown

a23blvlm

The directory server is running.

The directory server is currently running. To stop the server, click the **Shutdown** button.

Related tasks:

- Properties - Edit server properties
- SSL - Configure server SSL (Secure Sockets Layer) properties providing encryption of data, and transport of public-key certificates
- Master/replica configuration - Configure this server to be a master or a replica

Figure 30. Starting the directory server

Testing the installation

By being able to configure and start the directory server, you have proved that the product has been installed correctly.

What we have achieved

If you have followed all these product installation instructions, you should now have installed working versions of the following products:

❑IBM Java Development Kit Version 1.1.7 for Windows
❑IBM HTTP Server Version 1.3.6.2
❑IBM DB2 Universal Database for Windows NT Version 6.1 Workgroup edition with Fixpack 2 (or Fixpack 1A)
❑IBM VisualAge for Java Version 3.02, Enterprise Edition
❑IBM Distributed Debugger Version 8.4
❑IBM WebSphere Application Server Version 3.02, Standard Edition
❑IBM WebSphere Studio Version 3.02
❑IBM SecureWay Directory Version 3.1.1

4 Servlets

In this chapter we introduce you to Java servlet concepts.

We provide an overview of the Java Servlet API, and discuss the servlet runtime environment and life-cycle. Servlet examples are provided which demonstrate basic to advanced servlet functionality. Finally, we discuss some common servlet interaction techniques, such as servlet filtering and chaining.

If you want to run the examples presented here, refer to Chapter 6, “WebSphere Application Server” on page 123, and to Chapter 7, “Development and testing with VisualAge for Java” on page 167. All the examples are provided on the Internet (see Appendix C, “Using the additional material” on page 417).

We recognize that there is an abundance of both online and printed documentation on this topic, and recommend that you refer to the Sun Java Servlet API Specification, `http://java.sun.com/products/servlet/`.

If you are already familiar with Java servlets, we suggest you still skim through this chapter. We present some concepts here that are built on in subsequent chapters. This will familiarize you with the naming conventions used, and provide some continuity in the reading.

Overview of Java servlets

Servlets are protocol and platform independent server-side software components, written in Java. They run inside a Java enabled server or application server, such as the WebSphere Application Server. Servlets are loaded and executed within the Java Virtual Machine (JVM) of the Web server or application server, in much the same way that applets are loaded and executed within the JVM of the Web client. Since servlets run inside the servers, however, they do not need a graphical user interface (GUI). In this sense, servlets are also faceless objects.

Servlets more closely resemble Common Gateway Interface (CGI) scripts or programs than applets in terms of functionality. As in CGI programs, servlets can respond to user events from an HTML request, and then dynamically construct an HTML response that is sent back to the client.

Servlet process flow

Servlets implement a common request/response paradigm for the handling of the messaging between the client and the server. The Java Servlet API defines a standard interface for the handling of these request and response messages between the client and server.

Figure 31 shows a high-level client-to-servlet process flow:

1. The client sends a request to the server.
2. The server sends the request information to the servlet.
3. The servlet builds a response and passes it to the server. That response is dynamically built, and the content of the response usually depends on the client's request. External resources may also be used.
4. The server sends the response back to the client.

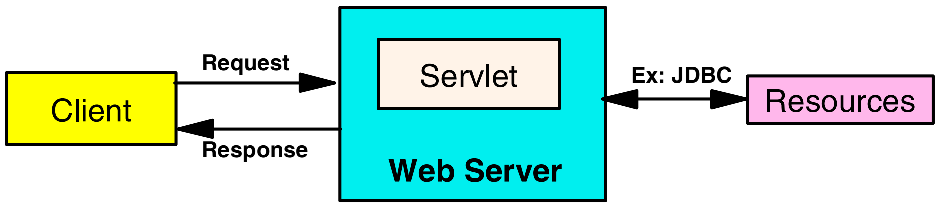


Figure 31. High-level client-to-servlet process flow

Servlets are powerful tools for implementing complex business application logic. Written in Java, servlets have access to the full set of Java API's, such as JDBC for accessing enterprise databases.

As mentioned above, servlets are similar to CGI in that they can produce dynamic Web content. Servlets, however, have the following advantages over traditional CGI programs:

❑*Portability and platform independence:* Servlets are written in Java, making them portable across platforms and across different Web servers, because the Java Servlet API defines a standard interface between a servlet and a Web server.

❑*Persistence and performance:* A servlet is loaded once by a Web server, and invoked for each client request. This means that the servlet can maintain system resources, like a database connection, between requests. Servlets don't incur the overhead of instantiating a new servlet with each request. CGI processes typically must be loaded with each invocation.

❑*Java based:* Because servlets are written in Java, they inherit all the benefits of the Java language, including a strong typed system, object-orientation, and modularity, to name a few.

The Java Servlet API

The Java Servlet API is a set of Java classes which define a standard interface between a Web client and a Web servlet. Client requests are made to the Web server, which then invokes the servlet to service the request through this interface.

The Java Servlet API is a Standard Java Extension API, meaning that it is not part of the core Java framework, but rather, is available as an add-on set of packages. We will be using the Java Servlet Development Kit API (JSDK) V2.1 conventions throughout this chapter.

The API is composed of two packages:

❑*javax.servlet*

❑*javax.servlet.http*

The *javax.servlet* package contains classes to support generic protocol-independent servlets. This means that servlets can be used for many protocols, for example, HTTP and FTP. The *javax.servlet.http* package extends the functionality of the base package to include specific support for the HTTP protocol. In this chapter, we will concentrate on the classes in the javax.servlet.http package.

The *Servlet* interface class is the central abstraction of the Java Servlet API. This class defines the methods which servlets must implement, including a *service()* method for the handling of requests. The *GenericServlet* class implements this interface, and defines a generic, protocol-independent servlet. To write an HTTP servlet for use on the Web, we will use an even more specialized class of *GenericServlet* called *HttpServlet*.

HttpServlet provides additional methods for the processing of HTTP requests such as GET (*doGet* method) and POST (*doPost* method). Although our servlets may implement a *service* method, in most cases we will implement the HTTP specific request handling methods of *doGet* and *doPost*.

The servlet life cycle

A client of a servlet-based application does not usually communicate directly with a servlet, but requests the servlet's services through a Web server or application server that invokes the servlet through the Java Servlet API. The server's role is to manage the loading and initialization of the servlet, the servicing of the request, and the unloading or destroying of the servlet. This is generally provided by a servlet manager function of the application server.

Typically, there is one instance of a particular servlet object at a time in the Web servers' environment. This is the underlying principle to the persistence of the servlet. The Web server is responsible for handling the initialization of this servlet when the servlet is first loaded into the environment, where it remains active (or persistent) for the life of the servlet.

Each client request to the servlet is handled via a new thread against the original instance object. The Web server is responsible for creating the new threads to handle the requests. The Web server is also responsible for the unloading or reloading of the servlets. This might happen when the Web application is brought down, or the underlying class file for the servlet changes, depending on the underlying implementation of the server.

Figure 32 shows a basic client-to-servlet interaction:

- ❑Servlet1 is initially loaded by the Web application server. Instance variables are initialized, and remain active (persistent) for the life of the servlet.
- ❑Two Web browser clients have requested the services of Servlet1. A handler thread is spawned by the server to handle each request. Each thread has access to the originally loaded instance variables that were initialized when the servlet was loaded.
- ❑Each thread handles its own requests, and responses are sent back to the calling client.

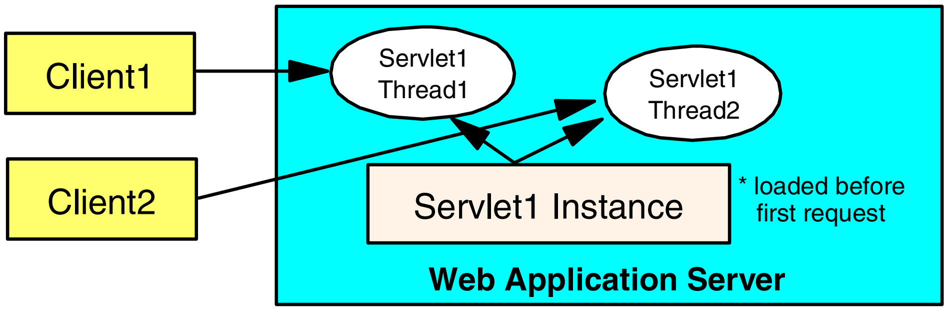


Figure 32. Basic client-to-servlet interaction

The life cycle of a servlet is expressed in the Java Servlet API in the *init*, *service* (*doGet* or *doPost*), and *destroy* methods of the Servlet interface. We will discuss the functions of these methods in more detail and the objects that they manipulate. Figure 33 is a visual diagram of the life-cycle of an individual servlet.

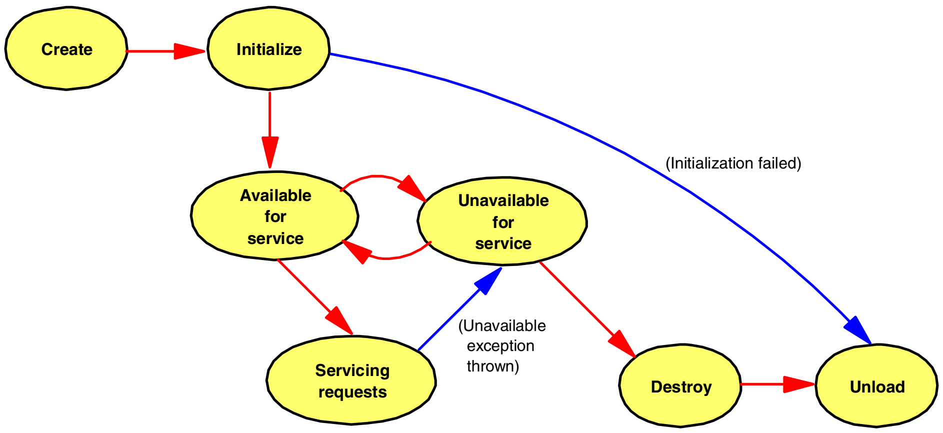


Figure 33. Servlet life-cycle

The WebSphere administrator can set an application and its servlets to be unavailable for service. In such cases, the application and servlets remain unavailable until the administrator changes them to available.

Understanding the life-cycle

This section describes in detail some of the important servlet life-cycle methods of the Java Servlet API.

Servlet Initialization: init method

Servlets can be dynamically loaded and instantiated when their services are first requested, or the Web server can be configured so that specific servlets are loaded and instantiated when the Web server initializes.

In either case, the *init* method of the servlet performs any necessary servlet initialization, and is guaranteed to be called once for each servlet instance, before any requests to the servlet are handled. An example of a task which may be performed in the *init* method is the loading of default data parameters or database connections.

The most common form of the *init* method of the servlet accepts a *ServletConfig* object parameter. This interface object allows the servlet to access name/value pairs of initialization parameters that are specific to that servlet. The *ServletConfig* object also gives us access to the *SevletContext* object that describes information about our servlet environment. Each of these objects will be discussed in more detail in the servlet examples sections.

Servlet request handling

Once the servlet has been properly initialized, it may handle requests (although it is possible that a loaded servlet may get no requests). Each request is represented by a *ServletRequest* object, and the corresponding response by a *ServletResponse* object in the Java Servlet API. Since we will be dealing with *HttpServlets*, we will deal exclusively with the more specialized *HttpServletRequest* and *HttpServletResponse* objects.

The *HttpServletRequest* object encapsulates information about the client request, including information about the client's environment and any data that may have been sent from the client to the servlet. The *HttpServletRequest* class contains methods for extracting this information from the request object.

The *HttpServletResponse* is often the dynamically generated response, for instance, an HTML page which is sent back to the client. It is often built with data from the *HttpServletRequest* object. In addition to an HTML page, a response object may also be an HTTP error response, or a redirection to another URL, servlet, or JavaServer Page. The redirection techniques will be discussed in more detail in the servlet interaction section of this chapter. JavaServer Pages and interactions with servlets will be discussed in Chapter 5, "JavaServer Pages" on page 95.

Each time a client request is made, a new servlet thread is spawned which services the request. In this way, the server can handle multiple concurrent requests to the same servlet. For each request, usually the *service, doGet,* or *doPost* methods will be called. These methods are passed the *HttpServletRequest* and *HttpServletResponse* parameter objects.

doPost: Invoked whenever an HTTP POST request is issued through an HTML form. The parameters associated with the POST request are communicated from the browser to the server as a separate HTTP request. The *doPost* method should be used whenever modifications on the server will take place.

doGet: Invoked whenever an HTTP GET method from a URL request is issued, or an HTML form. An HTTP GET method is the default when a URL is specified in a Web browser. In contrast to the *doPost* method, *doGet* should be used when no modifications will be made on the server, or when the parameters are not sensitive data. The parameters associated with a GET request are appended to the end of the URL, and are passed into the QueryString property of the *HttpServletRequest*.

Other servlet methods worth mentioning

destroy: The destroy method is called when the Web server unloads the servlet. A subclass of *HttpServlet* only needs to implement this method if it needs to perform cleanup operations, such as releasing database connections or closing files.

getServletConfig: The getServletConfig method returns a *ServletConfig* instance that can be used to return the initialization parameters and the *ServletContext* object.

getServletInfo: The getServletInfo method is a method that can provide information about the servlet, such as its author, version, and copyright. This method is generally overwritten to have it return a meaningful value for your application. By default, it returns an empty string.

Basic servlet examples

In this section, we will build on the foundation in the previous sections, by describing some servlets that demonstrate additional capabilities and concepts of the Java Servlet API.

Simple HTTP servlet

We begin with a look at a very simple servlet, *SimpleHttpServlet* (Figure 34).

```
package itso.servjsp.servletapi;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleHttpServlet extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                      throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
   out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>");
   out.println("<H4>This is about as simple a servlet as it gets!</H4>");
   out.println("</BODY><HTML>");
   out.close();
}
}
```

Figure 34. Simple HTTP servlet

As the title indicates, *SimpleHttpServlet* is a very simple HTTP servlet that accepts a request and writes a response. Let's break out the components of this servlet so we can discuss them individually.

Basic servlet structure

Figure 35 shows that we have defined this servlet to be part of an *itso.servjsp.servletapi* Java package. This is the naming convention used for all the servlet examples in this chapter.

```
package itso.servjsp.servletapi;
```

Figure 35. SimpleHttpServlet package declaration

Figure 36 shows the import statements used to give us access to other Java packages. The import of *java.io* is so that we have access to some standard IO classes. More importantly, the *javax.servlet.** and *javax.servlet.http.** import statements give us access to the Java Servlet API set of classes and interfaces.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Figure 36. SimpleHttpServlet import statements

Figure 37 shows the *SimpleHttpServlet* class declaration. We extend the *HttpServlet* class (*javax.servlet.http.HttpServlet*) to make our class an HTTP protocol servlet.

```
public class SimpleHttpServlet extends HttpServlet {
```

Figure 37. The SimpleHttpServlet class declaration

Figure 38 is the heart of this servlet, the implementation of the *service* method for the handling of the request and response objects of the servlet.

```
protected void service (HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
   out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>");
   out.println("<H4>This is about as simple a servlet as it gets!</H4>");
   out.println("</BODY><HTML>");
   out.close();
}
```

Figure 38. SimpleHttpServlet service method

What the service method does

Let’s examine this *service* method in more detail. Notice that the method accepts two parameters, *HttpServletRequest* and *HttpServletResponse*. The request object contains information about and from the client. In this example, we don’t do anything with the request.

This method is declared Abstract in the basic *GenericServlet* class, and so subclasses, such as *HttpServlet*, must override it. In our subclass of *HttpServlet*, when using this method, we must implement this method according to the signature defined in *HttpServlet*, namely, that it accepts *HttpServletRequest* and *HttpServletResponse* arguments.

We do some handling of the response object, which is responsible for sending our response back to the client. Our response here is a formatted HTML page, so we first set the response content type to text/html by coding **`res.setContentType("text/html")`**. Next, we request a *PrintWriter* object to write text to the response by coding **`PrintWriter out = res.getWriter()`**. We could also have used a *ServletOutputStream* object to write out our response, but *getWriter* gives us more flexibility with Internationalization. In either case, the content type of the response must be set before references to these objects can be made.

The remaining **`out.println`** statements write our HTML to the PrintWriter, which is sent back to the client as our response. It is pretty simple HTML, so we do not display it here. We use **`out.close`** more for completeness, because the Web application server automatically closes the PrintWriter when the *service* method exits.

How the servlet gets invoked

We could invoke this servlet with either a GET or POST form action method; the *service* method will execute for either. If we knew something about how this servlet was ultimately to be called, for instance, what the HTML form method was going to be, we could have implemented the above functionality through specific *doGet* or *doPost* methods. The result would be the same.

The simplest way to invoke the servlet would be by specifying a URL in the Web browser. This does not work for every servlet, but would work for the above example. A URL forces the Web browser to send the request using GET, similar to the way a standard HTML page is requested. The above servlet could be invoked from the Web browser with the URL:

```
http://host/servlet/itso.servjsp.servletapi.SimpleHttpServlet
http://host/itsoservjsp/servlet/itso.servjsp.servletapi.SimpleHttpServlet
```

Note: the second form invokes a servlet in a Web application.

Running the servlet

At this point we have not discussed the specifics of running servlets in a Web server environment. If you want to run this servlet, you should be able to follow the steps in Chapter 7, “Development and testing with VisualAge for Java” on page 167, code the *SimpleHttpServlet*, and run it under the WebSphere Test Environment. The WebSphere Test Environment provides a simulated Web server environment within the VisualAge for Java product and enables you to test and debug your servlets. Later, in Chapter 6, “WebSphere Application Server” on page 123, we discuss deploying servlets to the actual application server environment.

HTML form generator servlet

We next look at another simple HTTP servlet, *HTMLFormGenerator* (Figure 39).

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HTMLFormGenerator extends HttpServlet {

public void init(ServletConfig config) throws ServletException {
    super.init(config);
    System.out.println("In the init() method of HTMLFormGenerator");
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                                          throws ServletException, IOException {
    performTask(req, res, "POST",
                 "itso.servjsp.servletapi.HTMLFormHandler");
    //           "/itsoservjsp/servlet/itso.servjsp.servletapi.HTMLFormHandler");
}
public void performTask(HttpServletRequest req, HttpServletResponse res,
                     String method, String url) throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>HTMLFormGenerator</TITLE><BODY>");
    out.println("<H2>Servlet API Example - HTMLCreatingServlet</H2><HR>");
    out.println("<FORM METHOD=\"" + method + "\" ACTION=\"" + url + "\">");
    out.println("<H2>Tell us something about yourself: </H2>");
    out.println("<B>Enter your name: </B>");
    out.println("<INPUT TYPE=TEXT NAME=firstname><BR>");
    out.println("<B>Select your title: </B>");
    out.println("<SELECT NAME=title>");
    out.println("<OPTION VALUE=\"Web Developer\">Web Developer");
    out.println("<OPTION VALUE=\"Web Architect\">Web Architect");
    out.println("<OPTION VALUE=\"Other\">Other");
    out.println("</SELECT><BR>");
    out.println("<B>Which tools do you have experience with: </B><BR>");
    out.println("<INPUT TYPE=checkbox NAME=\"tools\"
          VALUE=\"WebSphere Application Server\">WebSphere Application Server<BR>");
    out.println("<INPUT TYPE=checkbox NAME=\"tools\"
          VALUE=\"WebSphere Studio\">WebSphere Studio<BR>");
    out.println("<INPUT TYPE=checkbox NAME=\"tools\"
          VALUE=\"VisualAge for Java\">VisualAge for Java<BR>");
    out.println("<INPUT TYPE=checkbox NAME=\"tools\"
          VALUE=\"IBM Http Web Server\">IBM Http Web Server<BR>");
    out.println("<INPUT TYPE=checkbox NAME=\"tools\" VALUE=\"DB2 UDB\">DB2 UDB<BR>");
    out.println("<INPUT TYPE=\"SUBMIT\" NAME=\"SENDPOST\" NAME=\"SENDPOST\">");
    out.println("</FORM>");

    out.println("</BODY><HTML>");
    out.close();
    System.out.println("In the doGet method");
}
}
```

Figure 39. HTML form generator servlet

init method

This servlet implements the *init* method. The init method only prints a message to standard output and call the super-class constructor. As we mentioned before, the *init* method is called only once, when the servlet is loaded. This message, therefore, should only be printed to the Web server's console or log once (wherever standard output is defined), regardless of how many times the servlet is actually invoked.

doGet method

We decided that this servlet is always called through a GET request, we have chosen to implement the *doGet* method, instead of the more generic *service* method. We developed a *performTask* method to which we pass a method posting type and a target URL.

Response object

The HTML page that this servlet generates is a bit more complex than the previous example. It actually builds an HTML form that can be used in the future to call other servlets. This is not the same as a servlet calling a servlet, which is a server-side process, and is discussed in "Servlet interaction techniques" on page 73. Here, we are just using one servlet to generate the HTML back to the browser, so we can call our other example servlets, and we do not have to create separate HTML files for each servlet.

Notice that this servlet has many `out.println` statements. This is just the HTML that is written back to the browser. Despite the size of this servlet, it is still only doing one simple thing, writing HTML output.

Invoking the servlet

This servlet can be invoked directly by a URL command:

```
http://hostname/servlet/itso.servjsp.servletapi.HTMLFormGenerator
http://hostname/webappname/servlet/itso.xxxx <== with web application
```

Notice the output line for the form that this servlet generates in the performTask method:

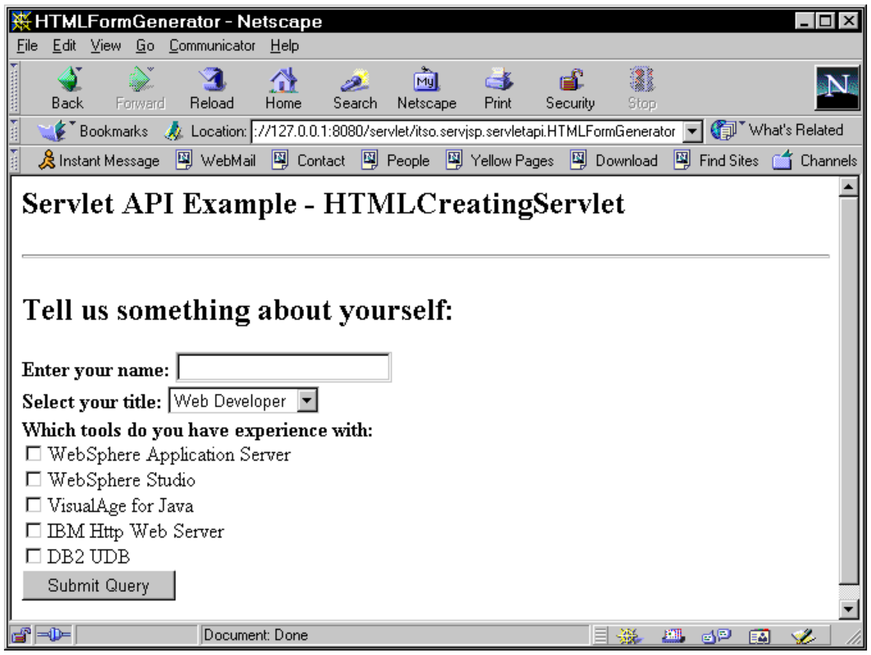
```
<FORM METHOD="POST"
    ACTION="itso.servjsp.servletapi.HTMLFormHandler">
```

This line demonstrates another way of invoking a servlet, in this case from a Web browser using a form action event. The form is generated by the *HTMLFormGenerator* servlet.

Note: The relative URL in the action is added to the current prefix of the generating servlet, such as `http://hostname/..../servlet/`.

Servlet output

The HTML Page that this servlet generates is shown in Figure 40.


HTMLFormGenerator - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: ://127.0.0.1:8080/servlet/itso.servjsp.servletapi.HTMLFormGenerator What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels
Document: Done

Servlet API Example - HTMLCreatingServlet

Tell us something about yourself:

Enter your name:

Select your title: Web Developer

Which tools do you have experience with:

☐ WebSphere Application Server
☐ WebSphere Studio
☐ VisualAge for Java
☐ IBM Http Web Server
☐ DB2 UDB

Submit Query

Figure 40. HTML form generator servlet: response output

HTML form processing servlet

We next look at a servlet that processes HTML form data. Figure 41 and Figure 42 show the *HTMLFormHandler* servlet.

```
package itso.servjsp.servletapi;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLFormHandler extends HttpServlet {

public void init (ServletConfig srvCfg) throws ServletException {
   super.init(srvCfg);
}
```

Figure 41. HTML form handler servlet (part 1)

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
                                               throws ServletException, IOException {
    res.setContentType("text/html");  //must be before first ref to PrintWriter
    PrintWriter out = res.getWriter();

    out.println("<HTML><TITLE>HTMLFormHandler</TITLE></BODY>");
    out.println("<H2>Servlet API Example - HTMLFormHandler</H2><HR>");

    //Retrieving the single-value parameters
    out.println("Hi <B>" + req.getParameter("firstname") + ",</B><P>");
    out.println("I see you are a <B>" + req.getParameter("title") + ",</B><P>");
    out.println("And have worked with the following tools: <BR>");
    //Retrieving the multi-value parameters
    String vals[] = (String []) req.getParameterValues("tools");
    if (vals != null) {
        for(int i = 0; i<vals.length; i++)
            out.println("<B>" + vals[i] + "</B><BR>");
    }
    else out.print;n("<B> None </B><BR>");

    out.println("<HR>");
    getReqInfo(req, out);                     //gets the standard request information

    out.println("</BODY></HTML>");
    out.close();
}

public void getReqInfo(HttpServletRequest req, PrintWriter out)
                                             throws ServletException, IOException {
    out.println("<H4><B>Additional Request Information:</B></H4>");
    out.println("<B>Request method:</B> " + req.getMethod() + "<BR>");
    out.println("<B>Request URI:</B> " + req.getRequestURI() + "<BR>");
    out.println("<B>Request protocol:</B> " + req.getProtocol() + "<BR>");
    out.println("<B>Request scheme:</B> " + req.getScheme() + "<BR>");
    out.println("<B>Servlet path:</B> " + req.getServletPath() + "<BR>");
    out.println("<B>Servlet name:</B> " + req.getServerName() + "<BR>");
    out.println("<B>Servlet port:</B> " + req.getServerPort() + "<BR>");
    out.println("<B>Path info:</B> " + req.getPathInfo() + "<BR>");
    out.println("<B>Path translated:</B> " + req.getPathTranslated() + "<BR>");
    out.println("<B>Character encoding:</B> "+req.getCharacterEncoding()+ "<BR>");
    out.println("<B>Query string:</B> " + req.getQueryString() + "<BR>");
    out.println("<B>Content length:</B> " + req.getContentLength() + "<BR>");
    out.println("<B>Content type:</B> " + req.getContentType() + "<BR>");
    out.println("<B>Remote user:</B> " + req.getRemoteUser() + "<BR>");
    out.println("<B>Remote address:</B> " + req.getRemoteAddr() + "<BR>");
    out.println("<B>Remote host:</B> " + req.getRemoteHost() + "<BR>");
    out.println("<B>Authorization scheme:</B> " + req.getAuthType() + "<BR>");
}
} //end of class
```

Figure 42. HTML form handler servlet (part 2)

Request object handling

So far, all of our servlet examples have only used the response object, but not the request object. This example shows how to process the data in the request. We assume that this servlet is always called using a POST request, and have therefore chosen to implement the d*oPost* request handling method.

doPost method

Incidentally, this servlet has been designed to handle the particular type of request from the HTML page that was generated in the previous servlet example. In that HTML page, the user could fill out information in the form and submit it. The action in the HTML form causes the *HTMLFormHandler* servlet to be invoked, and the *doPost* request handler method to be called:

```
<FORM METHOD="POST"
      ACTION="itso.servjsp.servletapi.HTMLFormHandler">
```

In the *doPost* method, we handle the *HttpServletRespons*e in the same way as before, except that this time, we are also handling the *HttpServletRequest*.

Getting form values

We use the *getParameter* method of the request to extract the values of the request parameters (name/value fields passed in from the HTML page). We extract parameters named *firstname* and *title* from the request:

```
req.getParameter("firstname")
req.getParameter("title")
```

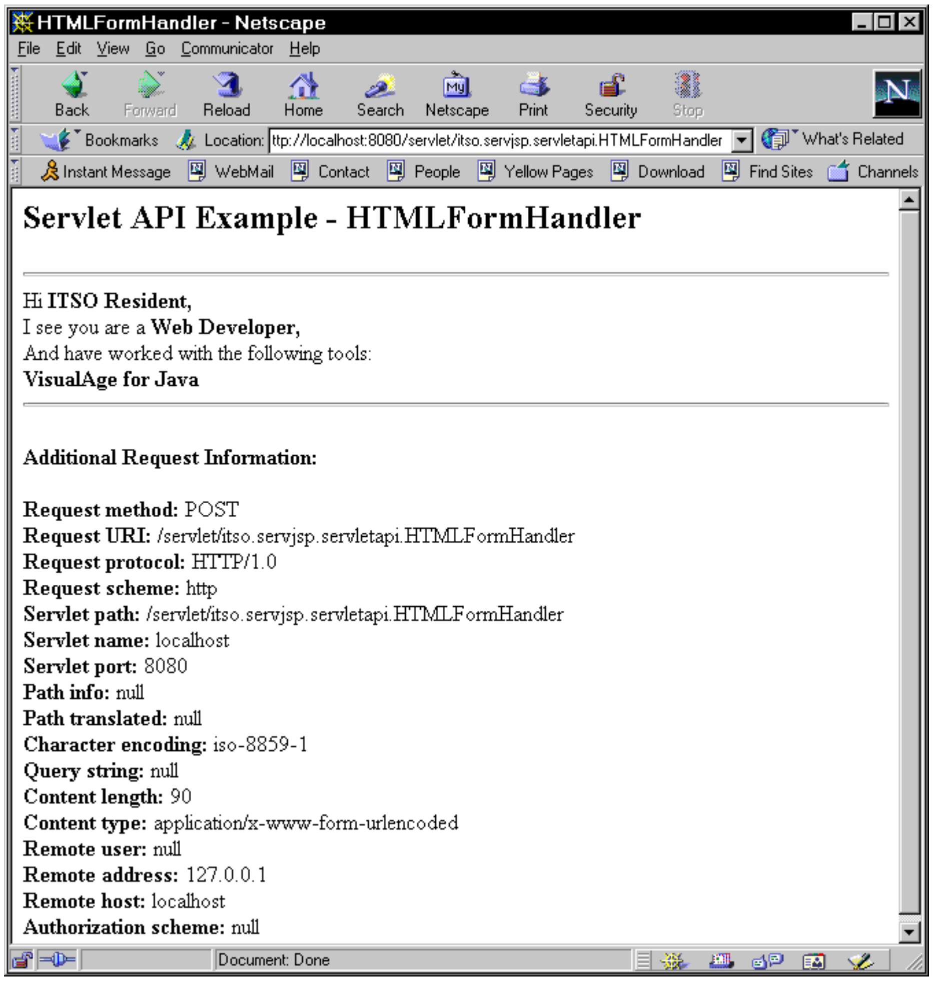
These are two of the input fields that were passed from the HTML form. The g*etParameter* method requires as an argument the name of the parameter that we want to extract (so it must be known), and returns the value of that parameter, or null. To get a list of the all parameter names, we could use the *getParameterNames* method. This method returns an enumeration of all the parameter names in the request, which we could then iterate through to get the individual parameter values.

To extract the value of the *tools* parameter, however, we must apply a slightly different technique. The tools' parameter is a multi-value input field (in this case, a checkbox). Because there could be more than one value to extract, we use the *getParameterValues* method, which returns an array of values.

General request properties

We can pull environment properties and other information about the client from the *HttpServletRequest* object and echo them to the response. We choose to put all this code in a separate method, *getReqInfo*, for ease of use.

The HTML page that this servlet generates is shown in Figure 43.


HTMLFormHandler - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: ttp://localhost:8080/servlet/itso.servjsp.servletapi.HTMLFormHandler What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels
Document: Done

Servlet API Example - HTMLFormHandler

Hi **ITSO Resident,**
I see you are a **Web Developer,**
And have worked with the following tools:
VisualAge for Java

Additional Request Information:

Request method: POST
Request URI: /servlet/itso.servjsp.servletapi.HTMLFormHandler
Request protocol: HTTP/1.0
Request scheme: http
Servlet path: /servlet/itso.servjsp.servletapi.HTMLFormHandler
Servlet name: localhost
Servlet port: 8080
Path info: null
Path translated: null
Character encoding: iso-8859-1
Query string: null
Content length: 90
Content type: application/x-www-form-urlencoded
Remote user: null
Remote address: 127.0.0.1
Remote host: localhost
Authorization scheme: null

Figure 43. HTML form handler servlet response output

Simple counter servlet

SimpleCounter is another simple servlet, but here we have an instance counter variable that is initialized in the init method (Figure 44).

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleCounter extends HttpServlet {

   private int calledCount;

public void init(ServletConfig config) throws ServletException {
      super.init(config);
      calledCount = 0;
}
protected void service(HttpServletRequest req, HttpServletResponse res)
                                 throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><BODY>");
   out.println("<H2>Servlet API Example - SimpleCounter</H2><HR>");
   ++calledCount;
   out.println("<H4>This servlet has been called: " + calledCount +
             " times.</H4>");
   out.println("</BODY><HTML>");
   out.close();
}
}
```

Figure 44. Simple counter servlet

Every time this servlet is invoked, we increment this counter variable, *calledCount*, by one. The first time this servlet loads, we initialize the counter to 0. Subsequent invocations keep incrementing the counter.

Persistence

This example demonstrates the *persistence* property of servlets, where an instance variable can remain active for the life of the servlet. Every time a servlet thread is spawned to handle the servlet request, it has access to this global instance variable. This could be useful in the case where these instance variables take a long time to initialize, such as database connections, and we want to set them once and reuse them with each invocation, without having to incur the initialization overhead each time.

This is a commonly used technique, particularly when we are only initializing data, and then reading global variables, as is the case with database connections. In this example, however, we are reading and updating this global variable. This introduces some issues that you have to consider when designing your servlets.

Multi-Threaded

Because our requests to this servlet are handled in threads against the same servlet object, we must implement mechanisms to guarantee thread safety for these shared instance variables, because we can update them in separate threads. In other words, there is no guarantee that the line that increments the counter and the line that prints out the counter will be executed asynchronously within a thread. So, we must identify critical sections of code, and synchronize these sections if appropriate.

There are many books that deal with concurrent programming issues, therefore we do not describe how to do this, but it is an important point to remember when designing your servlets. Please refer to Appendix E, “Related publications” on page 433 for a list of useful references.

Servlet initialization parameters

The *SimpleInitServlet* servlet shows how to retrieve initialization parameters from the servlet configuration object (Figure 45).

ServletConfig object

The *ServletConfig* object is a parameter that can be passed into the *init* method of the servlet. You can also get the *ServletConfig* object from the request object through the *getServletConfig* method, but it is most commonly used in the *init* method to initialize the servlet’s instance variables.

Methods of the *ServletConfig* object allow us to extract the parameter information from this object. This parameter information is in a name/value pairs format, and can be stored in a file in XML format. We do not have to read the file, however, because the methods of the class provide us with some handy helper methods.

What this servlet does

This servlet simply extracts the parameter information from the configuration file, and stores those values in instance variables. It then echoes this information back to the client that invoked the servlet. In a real-life application, these variables would most likely be used to make a connection to the database, and this connection would be stored in a global instance variable for later use in the *doGet* method.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleInitServlet extends HttpServlet {

   protected String mydriver;
   protected String myurl;
   protected String myuserID;
   protected String mypassword;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   mydriver = config.getInitParameter("driver");
   myurl = config.getInitParameter("URL");
   myuserID = config.getInitParameter("userID");
   mypassword = config.getInitParameter("password");
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();
   out.println("<HTML>");
   out.println("<TITLE>Date Display</TITLE>");
   out.println("<BODY>");
   out.println("<H2>Servlet Initialization Parameters (ServletConfig):
            </H2><HR>");
   out.println("<B>driver: </B>" + mydriver + "</BR>");
   out.println("<B>url: </B>" + myurl + "</BR>");
   out.println("<B>password: </B>" + mypassword + "</BR>");
   out.println("<B>userID: </B>" + myuserID + "</BR>");
   out.println("</BODY></HTML>");
   out.close();
}
}
```

Figure 45. Simple initialization servlet source: ServletConfig parameters

Servlet configuration file

The statement `mydriver = config.getInitParameter("driver")` extracts the *driver* parameter by name from the configuration file, and stores it in a global instance variable. The parameter information itself is actually stored in XML format, in a file called *SimpleInitServlet.servlet*. This file must be found through the class path. Where this file actually exists depends on your application server implementation, and is discussed in “Testing the servlets and JSPs” on page 423.

For VisualAge for Java testing, the file can be put into `d:\IBMVJava\ide\project_resources\..yourproject..\itso\servjsp\servletapi`.

The XML configuration file used in this example is shown in Figure 46. Here we have specified four parameters, for demonstration purposes only. These could be used to make a connection to a database.

```
<?xml version="1.0"?>
<servlet>
    <code>itso.servjsp.servletapi.SimpleInitServlet</code>
    <init-parameter value="COM.ibm.db2.jdbc.app.DB2Driver" name="driver"/>
    <init-parameter value="itso" name="password"/>
    <init-parameter value="jdbc:db2:sample" name="URL"/>
    <init-parameter value="itso" name="userID"/>
</servlet>
```

Figure 46. Servlet configuration file for simple initialization servlet

Understanding the configuration file format

Figure 47 shows the XML format of a configuration file. The WebSphere Application Server supports XML configuration files in this format.

```
<?xml version="1.0"?>
<servlet>
  <code>itso.servjsp.servletapi.SimplePageListServlet</code>
  <description>Shows how to use PageListServlet class</description>
  <init-parameter name="name1" value="value1"/>
  <page-list>
    <default-page>
      <uri>/index.jsp</uri>
    </default-page>
    <error-page>
      <uri>/error.jsp</uri>
    </error-page>
    <page>
      <uri>/itso/OutputA.jsp</uri>
      <page-name>pageA</page-name>
    </page>
    <page>
      <uri>itso/OutputB.jsp</uri>
      <page-name>pageB</page-name>
    </page>
  </page-list>
</servlet>
```

Figure 47. General XML configuration file format

Some of the parameters are beyond the scope of this section, however, we describe a few of the more important parameters that you should know. The elements (also known as tags) are:

- ❑*servlet:* The root element. The XMLServletConfig class automatically generates this element.
- ❑*code:* The class name of the servlet (without the .class extension), even if the servlet is in a JAR file.
- ❑*init-parameter:* The attributes of this element specify a name/value pair to be used as an initialization parameter. A servlet can have multiple initialization parameters, each within its own init-parameter element.
- ❑*page-list:* The elements within this tag specify JavaServer Pages that may be called by the servlet.

HTTP request handling utility servlet

We next look at a servlet, *ServletEnvironmentSnoop*. Because the source for this servlet is rather large, we have chosen to include it in Appendix B, "Utility servlet and utility JSP" on page 407.

This is a good utility servlet that extracts a lot of information from the request, and echoes its contents back to the client in the response. You should spend some time looking through the source code to see what kind of data can be extracted from a request object, and how to manipulate that data. Use this servlet as a future reference. Sample output of this servlet is also included in the appendix.

The *ServletEnvironmentSnoop* servlet demonstrates the handling of the following request data:

- ❑Request information—HTTP specific request information
- ❑Request header— data passed in the header of the request, such as the character and encoding sets
- ❑Request parameters—name/value pairs of parameter data
- ❑Request attribute names—attributes of the class
- ❑Request cookies—an array containing all cookies present in the request
- ❑Servlet configuration—values used for initializing the servlet
- ❑Servlet context attributes—information about the environment where the application server is running
- ❑Session information—session data associated with the request

Additional servlet examples

Now that we have covered some servlet basics, we will demonstrate some additional servlet techniques. We do not go into painstaking detail about how some of these work, and you can research the details by reading a more comprehensive book on servlets. Instead, we will focus on the important concepts each servlet demonstrates.

Cookie servlet

A cookie is a piece of data passed between a Web server and a Web browser. The Web server sends a cookie that contains data it requires the next time the browser accesses the server. This is one way to maintain state between a browser and a server.

The *CookieServlet* (Figure 48) demonstrates a servlet which gets and sets a cookie stored at a client. Initially, the browser may not have sent the cookie as part of the request, (for example, the first time it is called), so we just initialize a local *calledCount* variable to 0. If we are able to get this cookie from the request, we set the local calledCount to the value of the cookie.

The servlet first tries to get the calledCount by iterating through the cookies it received as part of the request. If no cookie contains the calledCount item, then the servlet initializes the calledCount value to 0. This value is then incremented, and a new cookie instance is created for calledCount and added to the response.

If we call this servlet from a URL, we find that the first time we call it, the calledCount is 0. Subsequent calls to the same servlet from this Web browser will show that we keep incrementing the counter, and storing it into the cookie sent back to the browser.

This is one way by which we can maintain state between the Web browser and the server. The major drawback with cookies is that most browsers enable the user at the client machine to deactivate (not accept) cookies.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CookieServlet extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   int calledCount = 0;
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>CookiesServlet</TITLE><BODY>");
   out.println("<H2>Servlet Cookie Example:</H2><HR>");
   if (getReqCookie(req, out, "calledCount") == null)
      calledCount = 0;
   else
      calledCount = new Integer(getReqCookie(req, out,
                             "calledCount")).intValue();
   out.print("The value of the cookie calledCount sent in on the request: ");
   if (calledCount == 0) out.println
                 ("null - value not sent in on request<HR>");
   else out.println(calledCount + "<HR>");
   calledCount++;
   Cookie cookie = new Cookie("calledCount",
                           new Integer(calledCount).toString());
   res.addCookie(cookie);
   out.println("The value of the cookie calledCount set on the response: " +
              calledCount);
   out.println("</BODY><HTML>");
   out.close();
}
private String getReqCookie(HttpServletRequest req, PrintWriter out,
                           String name) {
   Cookie[] cookies = req.getCookies();
   if (cookies != null && cookies.length > 0) {
      for(int i=0; i<cookies.length; i++) {
         if (cookies[i].getName().equals(name))
             return (cookies[i].getValue());
      }
   }
   return null;
}
}
```

Figure 48. Cookie servlet: state tracking using cookies

URL rewriting servlet

URL rewriting is another way to support state tracking. With URL rewriting, the parameter that we want to pass back and forth between the Web browser and client is appended to the URL. URL rewriting is the lowest common denominator of session tracking, and is used when a client does not accept cookies. We modified the *CookieServlet* to implement the same state tracking mechanism technique, but by using URL rewriting. The *URLServlet* (Figure 49) demonstrates this technique.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class URLServlet extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                    throws ServletException, IOException {
   int calledCount = 0;
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>URLServlet</TITLE><BODY>");
   out.println("<H2>Servlet URL Rewriting Example:</H2><HR>");
   calledCount = getReqURLInt(req, “calledCount”);
   out.println("The value of the url-parm calledCount received in the
               request:");
   if (calledCount == 0) out.println("null - value not received <HR>");
   else out.println(calledCount + "<HR>");
   calledCount++;
   out.println("The value of the url-parm calledCount sent back: " +
               calledCount);
   out.print("<HR><P><A HREF=\"itso.servjsp.servletapi.URLServlet");
   out.print("?calledCount=" + calledCount +
            "\"> Click to reload</A>");
   out.println("</BODY><HTML>");
   out.close();
}
public int getReqURLInt(HttpServletRequest req, String name) {
   int val = 0;
   if (req.getParameter(name) != null)
      val = new Integer(req.getParameter(name)).intValue();
   return val;
}
}
```

Figure 49. URL servlet: state tracking using URL rewriting

A real persistent servlet — between servlet life-cycle

In the *SimpleCounter* servlet (Figure 44 on page 57) we introduced a servlet that incremented a counter with every request to the servlet. We wanted to demonstrate that the servlet is persistent between requests.

The problem

If the server is brought down, however, the servlet would be reloaded, and the counter set back to zero. What if we wanted to store this counter between servlet life-cycle sessions? Every time the servlet initializes, we want to be able to reset it to the value of the last servlet life-cycle session.

A solution

We could do this by storing the counter variable in a file, and then loading this file into the counter variable in the next initialization. In this way, we have persistence between servlet life-cycle sessions.

The *PersistentCounter* servlet demonstrates how we might do this. We create our own object type, *SaveServletStats*, with a *calledCount* variable. We make the *SaveServletStats* object *Serializable*, so that we can save it to an *ObjectOutputStream* file (we use an object here because serialization is not supported for native data types, such as *int*).

The *init* method gets the file name of the stats file from the *ServletConfig*, then rebuilds the *SaveServletStats* object from the serialized file by using the *ObjectInputStream*. Once the *SaveServletStats* object has been rebuilt, we now have restored the *calledCount* value from our last servlet life-cycle session. If the file does not exist, we initialize it for the first time to zero. The PersistentCounter.servlet file is shown in Figure 50.

```
<?xml version="1.0"?>
<servlet>
     <code>itso.servjsp.servletapi.PersistentCounter</code>
     <init-parameter value="statsfile" name="filename"/>
</servlet>
```

Figure 50. Servlet configuration file for persistent counter servlet

The *PersistentCounter* servlet is shown in Figure 51. In the *doGet* method we save the file after each invocation (which would slow down performance slightly). To be thread safe, we synchronized this block. We could have put saving the file in the *destroy* method, but if the server crashed, we would not have the interim values.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersistentCounter extends HttpServlet {
   private int calledCount;
   private SaveServletStats stats;
   private String filename;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   calledCount = 0;
   filename = config.getInitParameter("filename");
   stats = new SaveServletStats();
   if (filename != null) {
      try { ObjectInputStream in = new ObjectInputStream(
                  new FileInputStream(filename  + ".ser"));
         stats = (SaveServletStats) in.readObject();
         in.close(); }
      catch (Exception e) { e.printStackTrace(); }
   }
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                                  throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();
   calledCount++;
   out.println("<HTML><TITLE>PersistentCounter</TITLE><BODY>");
   out.println("<H4>This servlet has been called: </H4><BR>");
   out.println("<B>" + calledCount + "</B> times since the servlet was loaded
                THIS servlet life-cycle session<BR>");
   out.println("<B>" + stats.calledCount + "</B> times since the servlet was
               loaded ALL servlet life-cycle sessions<BR>");
   out.println("</BODY></HTML>");
   stats.calledCount++;
   synchronized (this) {
      if (filename != null) {
         ObjectOutputStream outstats = new ObjectOutputStream(
                            new FileOutputStream(filename + ".ser"));
         outstats.writeObject(stats);
         System.out.println("Saving stats file: " + stats.calledCount);
         outstats.close(); } }
   out.close();
} }
```

Figure 51. Persistent counter servlet: state tracking in a file

Synchronizing access to our instance variables slows down the performance of this servlet, but it guarantees that only one thread can update the data at a time. This trade-off between servlet performance and data integrity is a common issue you must deal with when designing servlets.

The object that we are saving in serialized format is of type S*aveServletStats* (Figure 52).

```
package itso.servjsp.servletapi;
import java.io.*;
public class SaveServletStats implements Serializable {
   public int calledCount = 0;
}
```

Figure 52. SaveServletStats: serialized object

The file that stores the serialized object is written to the directory:

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment       <== VA Java
c:\Winnt\system32                                                      <== WebSphere
```

You can delete this file to restart the counter at zero.

User sessions

We have introduced several approaches to session and state tracking between Web browsers and the Web server. One limitation with our first two counter servlet examples, *SimpleCounter (*Figure 44 on page 57) and *PersistentCounter* (Figure 51 on page 66), is that they maintain the counter variable globally, within the servlet session, not by user.

We also showed a couple of session tracking mechanisms at the user level, *CookieServlet* (Figure 48 on page 63) and *URLServlet* (Figure 49 on page 64), where we maintain the counter variable per user, between multiple requests from the same Web browser to the Web server. In these methods, the developer is responsible for manually managing all of the session information within the code.

HttpSession

Luckily, the Java Servlet API has a class, *HttpSession*, which supports built-in session tracking between the client and the server, by user. HTTP is, by design, a stateless protocol. The *HttpSession* interface allows a server to use several approaches to track a user's session, or state, and makes it easy for the developer to use. The session information is managed at the user level.

The Java Servlet API supports two ways to associate multiple requests with a session: URL rewriting and cookies. In either case, the implementation details in the servlet are the same. A unique *session ID* is used to track multiple requests from the same client to the server, and this is what is passed as the URL or cookie parameter. The actual session object that we are tracking is maintained on the server.

Cookies

Session tracking through HTTP cookies is the most commonly used session tracking mechanism. In this way, the servlet container sends a cookie to the client, and the client will return the cookie on each subsequent request. The name of the session tracking cookie is JSESSIONID.

Although it is sent as a cookie, you as the developer do not need to manipulate it as such; the *HttpSession* class does all that for you.

Using HttpSession

Using *HttpSession* makes it easy for the developer to maintain and access session information within a servlet. It associates an HTTP client with an HTTP session, and it persists over multiple connections by the same user.

User session counter servlet

The *UserSessionCounter* servlet (Figure 53) demonstrates how to keep a session counter by user, using the *HttpSession* tracking technique.

The flow of this servlet can be described in the following steps:

❑We get a handle to a session object using the *getSession* method of the request. This method returns the current valid session associated with this request and user. This method takes a boolean argument, true means a new session should be created if none exists, false only returns an existing session, or null.

❑If it is a new session, or if the session does not contain our object, we must add an object into the session of the type that we want to keep around, using the *putValue* method of *HttpSession*. In this case, the *SaveServletStats* object (Figure 52 on page 67) contains the counter variable.

❑We now have to create a reference to the *SaveServletStats* object in the servlet. We use the *getValue* method of *HttpServlet* to retrieve this object.

Once you have a reference to your object through *getValue*, you can just manipulate the object as needed; updates to the object are automatically stored as part of the session object.

```
package itso.servjsp.servletapi;
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserSessionCounter extends HttpServlet {
   private int calledCount;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   calledCount = 0;
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();

   HttpSession session = req.getSession(true);
   if (session.isNew() || session.getValue("usersession")==null) {
      session.putValue("usersession", new SaveServletStats());
   }
   SaveServletStats ustats =
            (SaveServletStats)session.getValue("usersession");
   calledCount++;
   ustats.calledCount++;
   out.println("<HTML><TITLE>SessionCounter</TITLE><BODY>");
   out.println("<H4>This servlet has been called: </H4><BR>");
   out.println("<B>" + calledCount + "</B> times since the servlet was loaded
             THIS servlet life-cycle session<BR>");
   out.println("<B>" + ustats.calledCount + "</B> times since the servlet was
             loaded by this user<BR>");
   out.println("</BODY></HTML>");
   out.close();
}
}
```

Figure 53. User session servlet: state tracking by user

Session object types

As you can see, you can store different object types in a session, distinguished by name. We used the S*aveServletStats* class (Figure 52 on page 67) as a session object.

JDBC servlet

In *JDBCInitServlet* (Figure 54 and Figure 55) we extend the *SimpleInitServlet* example of Figure 45 on page 59 to actually make a connection to a DB2 database from the variables we initialize from the servlet configuration file. This example demonstrates how to make a connection to an external resource, and print the results back in the response.

```
package itso.servjsp.servletapi;
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class JDBCInitServlet extends SimpleInitServlet {
   protected Connection conn = null;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   try {
      // load JDBC driver
      Class.forName(mydriver).newInstance();
      conn = DriverManager.getConnection(myurl, myuserID, mypassword);
      System.out.println("Connection successful..");
   }
   catch (SQLException se) { System.out.println(se); }
   catch (Exception e)     { e.printStackTrace(); }
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();
   out.println("<HTML>");
   out.println("<TITLE>JDBC Init Connection</TITLE>");
   out.println("<BODY>");
   try { executeSQL(out); }
   catch (SQLException se) { se.printStackTrace(); }
   out.println("</BODY></HTML>");
   out.close();
}
```

Figure 54. JDBC servlet: part 1 — connecting to a JDBC database

```
public void executeSQL(PrintWriter out) throws SQLException{
   Statement stmt = conn.createStatement();
   String sql = "SELECT * FROM DEPARTMENT";
   stmt.executeQuery(sql);
   ResultSet rs = stmt.getResultSet();
   int count = 1;
   while (rs.next()) {
      out.println("<B>"+rs.getString("DEPTNAME")+"</B><BR><BLOCKQUOTE>");
      String sql2 = "SELECT * FROM EMPLOYEE WHERE WORKDEPT = '" +
                   rs.getString("DEPTNO") + "'";
      Statement stmt2 = conn.createStatement();
      stmt2.executeQuery(sql2);
      ResultSet rs2 = stmt2.getResultSet();
      while(rs2.next()) {
         out.println(rs2.getString("FIRSTNME") + " " +
                    rs2.getString("LASTNAME") + "<br>");
      }
      out.println("</BLOCKQUOTE>");
   }
}
}
```

Figure 55. JDBC servlet: part 2 — SQL access

The *JDBCInitServlet* extends the *SimpleInitServlet*. This demonstrates that we can consider designing base servlet classes at an application level and then extend them for a specific function. This servlet was built primarily to demonstrate functionality; exception handling has been left out in order to keep the code concise.

We choose to extend the *SimpleInitServlet*, and override the *doGet* method to make our processing specific to this example. The *executeSQL* method performs the actual SQL database calls.

This servlet connects to the SAMPLE database that is installed with DB2. It must first load up the database driver and make the connection. In this case, we choose to make the connection object (`Connection conn`) a shared instance variable, which we reuse from servlet request to servlet request, but initialize only once.

The connection information (user ID, password, URL, and driver) is specified in the JDBCInitServlet.servlet file, which must be copied to the appropriate directory.

Servlet tag with SHTML

With the *SHTMLServlet*, we demonstrate how to make the Web application server dynamically generate part of its HTML file. Using the servlet tag technique, the server converts a section of an HTML file into a dynamic portion each time the document is sent to the client. This dynamic portion invokes an appropriate servlet, and inserts the response of that server in the HTML page that is sent to the Web client. Initialization and other servlet parameters can be passed through the tag syntax, similar to the way an applet's parameters are set in the HTML. Here, the Web browser is calling the servlet indirectly, through the SHTML page. The Web server is responsible for including the output of the specified servlet in the HTML response.

The HTML syntax is as follows:

```
<servlet> name="myServlet" code="package.classname" </servlet>
```

Figure 56 shows the HTML that we used to call our servlet, and Figure 57 shows the servlet whose response is dynamically included between the tags. Notice that this servlet only generates a part of the total response back to the Web client.

```
<HTML><BODY>
<H2>Start of SHTML Servlet Example, the following lines are from the servlet:
</H2> <HR>
<SERVLET name="SHTMLServlet”
        CODE="itso.servjsp.servletapi.SHTMLServlet"> </SERVLET>
<HR> <H2>ENd of servlet include</H2>
</HTML></BODY>
```

Figure 56. SHTML file: servlet include (SHTMLServlet.shtml)

The <SERVLET> tag has been replaced with <jsp:include> in JSP 1.0. See "Calling a servlet from a JSP" on page 107 for examples of how to invoke a servlet from a JSP, which is a more modern technique to accomplish the same purpose.

To get this example working in WebSphere and VisualAge for Java, you must associate the .shtml extension with the JSP 0.91 compiler (it does not work with JSP 1.0). See "Additional servlet examples" on page 424 for instructions.

Because this technique is not commonly used, we do not elaborate further on this technique, but rather focus on JavaServer Pages (JSPs) and other server-side techniques in the chapters that follow.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SHTMLServlet extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res)
                                    throws ServletException, IOException {
   PrintWriter out = res.getWriter();
   out.println("<HR><H4>Servlet API Example - SHTMLServlet</H4>");
   out.println("<H4>Basic included servlet...</H4><HR>");
}
}
```

Figure 57. SHTML servlet: included servlet

Servlet interaction techniques

In our examples so far, we have demonstrated various servlet concepts and techniques. In most cases, these examples consisted of stand-alone servlet programs that handled a request and returned a response.

In the real world, servlets would not be stand-alone programs, but rather, they would be grouped together as part of an application; and the application components could consist of servlets, shared objects, and other resource files, such as HTML and JSPs. We call this our Web application in the WebSphere Application Server environment.

It would be expected that the servlets of an application would need some way to communicate and interact, either with each other or with the resources of the application. The *ServletContext* object, which we describe in more detail in “Application level scope” on page 89, provides a way for us to define this Web application level, and the resources that it can access and interact with.

This section describes various techniques for servlet interaction and communication. We will discuss the following types:

❑ *Servlet collaboration:* Two techniques for servlet collaboration are servlet filtering and chaining. Here multiple servlets collaborate on producing a single response for a client. The servlets themselves are not really interacting directly with each other, rather, the Web application server is responsible for tying the servlets together.

- ❑*Calling servlets from servlets:* Since servlets are Java programs, they can do anything a standard Java program can do, such as make a network connection. In this way, we have implicit servlet interaction. Additionally, because a servlet is just a Java class, we can instantiate and call a servlet's public methods.
- ❑*Response redirection:* We can redirect the servlet response to another application resource, such as another servlet or an error page (HTML or JSP). We discuss JSPs, and their interactions with servlets, in detail in Chapter 5, "JavaServer Pages" on page 95. They are just mentioned here as another servlet resource.
- ❑*Request dispatching:* Through the *RequestDispatcher* object, we can forward a request to another servlet, which can handle the request and return the response. Additionally, we can include directly another servlet's response within the context of a calling servlet. We can use request dispatching to dispatch the handling to another active application resource.
- ❑*Resource usage:* We can interact with an application's resources through the servlet context. The *ServletContext* object allows us access to these resources through the *getResource* method.
- ❑*Sharing of objects in scope:* There are three levels of object scope for a servlet. Application scope is between all servlets in the same application, and is accessed through the *ServletContext* object. User session objects are accessed through the *HttpSession* object, and request level objects through the servlet request.

We will provide a more detailed discussion, and some examples for each of these techniques.

Servlet collaboration: filtering and chaining

If multiple servlets are needed to produce a response to a particular client, then the normal procedure for producing HTML responses becomes a little more complex. Two ways for multiple servlets to collaborate on the response are filtering and chaining. We will discuss how these techniques work in the WebSphere Application Server environment.

Note: We recognize that sometimes the terms filtering and chaining are used interchangeably. However, in our discussion here, we use filtering to refer to only the MIME type filtering.

Servlet MIME filtering

In servlet filtering, the servlet changes the MIME type of the response it sends from *text/html* to a user-defined MIME type. When using text/html, the Web application server would normally send the response straight back to the browser. With our own MIME type, we configure the Web application server to associate the MIME type with a particular servlet, and the output of the first servlet is used as input to the second servlet. In this way, servlets can filter their output as input to other servlets. Figure 58 shows the servlet filtering process flow.

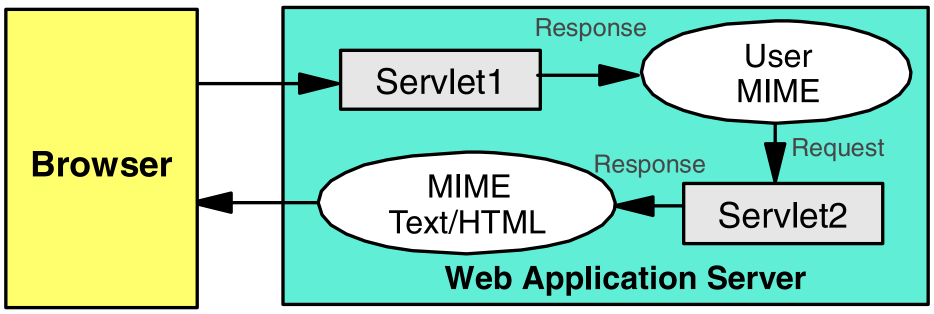


Figure 58. Servlet filtering process flow

In taking the output of one servlet, and using it as input to another servlet, this is useful for translation or substitution, for example, if you want to convert from the XML of one servlet into HTML for the user. Servlet filtering may have to be explicitly enabled in the Web Application Server through the `httpd.properties`, `enable.filters=true` property.

See “Servlet interaction techniques” on page 426 for instructions on how to set up this example.

Figure 59 shows how we can write to a specially defined MIME type from one servlet. On the server, we define a second servlet to be the handler of this MIME type (Figure 60).

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FilterFirst extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   res.setContentType("text/Deb");
   PrintWriter out = res.getWriter();
   out.println("<H2>Servlet API Example - FilterFirst</H2><HR>");
   out.println("<H4>Output from the FilterFirst servlet</H4>");
   out.close();
}
}
```

Figure 59. Servlet filtering example — MIME caller

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FilterSecond extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                    throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   //reading the output from the first servlet..
   BufferedReader in = req.getReader();
   String line;
   out.println("<HTML><BODY>");
   while((line = in.readLine()) != null)
      out.println(line);
   out.println("<H4><font color=\"Green\">This part of the output produced
                by the second filter servlet..</H4>");
   out.println("</BODY></HTML>");
   out.close();
}
}
```

Figure 60. Servlet filtering example — MIME handler

Servlet chaining

In servlet chaining, multiple servlets are called for a single client HTTP request, each servlet providing part of the HTML output. Each servlet receives the original client HTTP request as input, and each servlet produces its own output independently. Figure 61 shows the servlet chaining process flow.

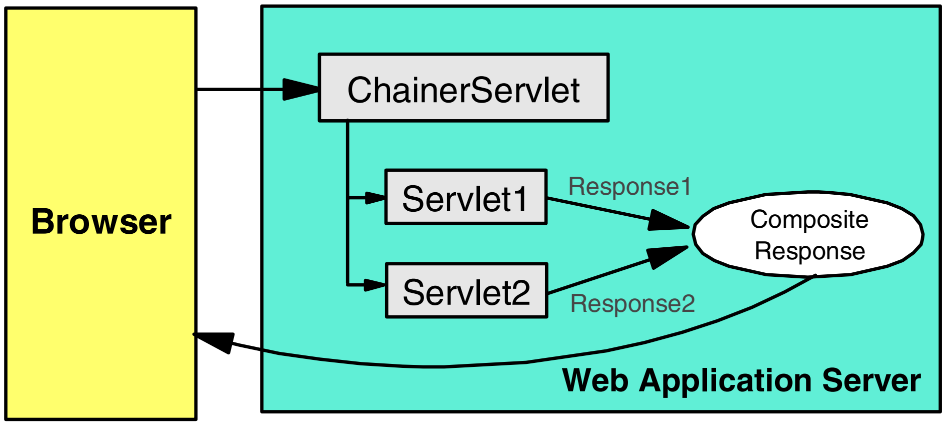


Figure 61. Servlet chaining process flow

WebSphere provides a *ChainerServlet* (in com.ibm.websphere.servlet.filter) that is invoked through a servlet alias. This servlet is specified on the original request, and multiple servlets are specified in an initialization parameter as the target:

```
Parameter name:  chainer.pathlist
Parameter value: /chainFirst /chainSecond
```

Each servlet is called in the order specified on the alias, and the output HTML is made up of the output from all of the servlets.

Servlet filtering and chaining have the advantage of allowing the Web developer to create modular servlets that can, for example, output standard HTML headers and footers or provide common dynamic content for pages.

Figure 62 and Figure 63 show how two servlets can be used in collaboration to produce a single output response. Notice that the second servlet must process the output of the first servlet, in order to produce the desired composite result. One possible application of this technique might be when one servlet produces as its output an XML formatted response, and we chain it to another servlet that wraps the appropriate style sheet around the XML before sending it back to the browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ChainerFirst extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>ChainerFirst</TITLE><BODY>");
   out.println("<H2>Servlet API Example - ChainerFirst</H2><HR>");
   out.println("<H4><font color=\"Red\">This part of the output produced by
               the first servlet..</font></H4>");
}
}
```

Figure 62. Servlet chaining: first servlet in the chain process

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ChainerSecond extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   res.setContentType(req.getContentType());
   PrintWriter out = res.getWriter();
   //Need this to read through the output of the last servlet:
   BufferedReader in = req.getReader();
   String line;
   while((line = in.readLine()) != null)
      out.println(line);
   out.println("<H4><font color=\"Green\">This part of the output produced
                by the second servlet..</H4>");
   out.println("</BODY></HTML>");
   out.close();
}
}
```

Figure 63. Servlet chaining: second servlet in the chain process

See “Servlet interaction techniques” on page 426 for instructions on how to set up this example.

Calling servlets from servlets

By using standard features of the Java language, such as the built-in networking support of the *java.net* package, a servlet can access another servlet's resources in a number of different ways.

A servlet can make an HTTP request to another servlet, and filter this response back to the client. By opening a connection to a URL, an HTTP request can be made, and a response received. In this way, the servlet is acting as both a client and a server-side process, and it has access to resources that could be on another server. There is really no magic here, any Java program can theoretically perform this kind of function.

A servlet can also instantiate a servlet object, and call its public methods directly, if the called servlet class is found relative to the original servlet's scope (which is usually on the same server). To instantiate a servlet object, you can use these methods:

❑*Getting the object via servlet context:* Using the ServletContext object, we can get access to any other servlets that are part of the Web application that we defined:

```
myHttpServlet myServlet =
        getServletConfig().getServletContext().getServlet("myServlet");
```

❑*Using class instantiation:* This is just standard Java class instantiation, and we just have to find the servlet class in the class path of the Web application:

```
myHttpServlet myServlet =
        (myHttpServlet)Class.forName("myServlet").newInstance()
```

This technique was common in the JSDK 2.0. We will not show any examples of this technique because request dispatching, available in JSDK 2.1, is more preferred. It does, however, demonstrate one technique for servlet-to-servlet interaction.

Response redirection

We can redirect the response of the servlet to another application resource. This resource may be another servlet, an HTML page, or a JSP. The resource (URL) must be available to the calling servlet, in the same servlet context.

There are two forms of response redirection that we discuss:

❑*Sending a standard redirect:* Using a `response.sendRedirect("myHtml.html")` page sends the HTML page as the response. If this page were another servlet, that servlet *does not* have access to the original request object, but its response would be sent. To have access to the request object, you must use the request dispatching technique discussed below.

- *Sending a redirect to an error page:* Here, an error code is sent as a parameter of the method, as in `response.sendError(response.SC_NO_CONTENT)`. The error numbers are predefined constants of the response. The defined error page is displayed with the appropriate error message. What this page looks like is dependent on how this feature is configured for the application.

Figure 64 shows a servlet which redirects its response conditionally to either an error page, using the *sendError* method, or a standard HTML page, using the *sendRedirect* method, depending on the contents of the request. This servlet example is called from a form POST in an HTML page, generated by the servlet *HTMLFormGeneratorRedirect* (subclass of HTMLFormGenerator).

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLFormHandlerRedirect extends HTMLFormHandler {

public void doPost(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>HTMLFormHandler</TITLE></BODY>");
   out.println("<H2>Servlet API Example -
               HTMLFormHandlerRedirect</H2><HR>");
   out.println("Hi <B>" + req.getParameter("firstname") + ",</B><P>");
   String title = req.getParameter("title");
   if (title.equals("Web Architect"))
      res.sendError(res.SC_BAD_REQUEST, "Sorry, but Web architects can't
                    see the page");
   else res.sendRedirect("HTMLFormHandlerRedirect.html");
   out.println("And have worked with the following tools: <BR>");
   out.println("</BODY></HTML>");
}
}
```

Figure 64. Response redirection servlet: redirecting using two techniques

Request dispatching

When building a Web application, it is often useful to forward the processing of a request to another servlet, or to include the output of another servlet in the response. The *RequestDispatcher* interface provides a mechanism to accomplish this, by defining a request dispatcher object that receives a request from the client and sends it to any resource to be further processed.

Similar to response redirection, this resource may be another active server resource, such as a servlet or JSP file, and the resource (URL) must be available to the calling servlet, in the same servlet context. Unlike response redirection, however, the request object *is* available to the called resource, in other words, it remains in scope.

We define an active application resource as one which can handle the request, such as another servlet or a JSP file (something in this case which has access to the request object). A passive resource would be an HTML file, which cannot explicitly handle the request, but is available to the application, usually in the document root.

An object implementing the *RequestDispatcher* interface may be obtained by the *ServletContext* through the *getRequestDispatcher* method. This method takes a String argument describing a path within the scope of the *ServletContext*. The path must be relative to the root of the *ServletContext.*

There are two ways to use a request dispatcher object:

- ❑*RequestDispatcher.forward:* Forwards the responsibility of processing the request and creating the response to another active resource. It is illegal to use this method if a reference to the *PrintWriter* output object has already been made (which is responsible for sending the response). *HTMLFormHandlerDispatcher1* (Figure 65) calls *DispatcherForward* (Figure 66) using the *forward* method to hand off the responsibility of processing the request and sending the response from the calling servlet to the called servlet.
- ❑*RequestDispatcher.include:* The include method of the *RequestDispatcher* interface provides the calling servlet the ability to respond to the client, but to use the included object's resource for *part* of the reply. Here, it can have the *PrintWriter* output object open, because the calling servlet is still responsible for handling the request. The called resource, however, cannot set headers in the client response. *HTMLFormHandlerDispatcher2* (Figure 67) calls *DispatcherInclude* (Figure 68) where we are using these two servlets together to produce a single response. Control is returned to the calling servlet.

In both the *forward* and *include* methods, the request object remains in the scope of the called object. The primary point to remember here is that when using forward, you must handle all writing of the response in the called servlet. When using include, you can write in either, or both, but you can only set the headers in the calling servlet.

The *HTMLFormHandlerDispatcherX* servlets are invoked from an HTML form generated by appropriate *HTMLFormGeneratorDisplatcherX* servlets.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLFormHandlerDispatcher1 extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
    RequestDispatcher rd =  getServletContext().getRequestDispatcher
              ("/servlet/itso.servjsp.servletapi.DispatcherForward");
    rd.forward(req, res);
}
}
```

Figure 65. Request dispatching servlet: calling servlet through forward method

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DispatcherForward extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><BODY>Start of FORWARDED request");
   out.println("<P>Hi " + req.getParameter("firstname"));
   out.println("<BR>I see you are a " + req.getParameter("title"));
   out.println("<P>End of request</BODY></HTML>");
}
}
```

Figure 66. Request dispatching servlet: called servlet through forward method

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLFormHandlerDispatcher2 extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res)
                                  throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><BODY>Start of INCLUDED request");
   out.println("<P>Hi " + req.getParameter("firstname"));
   out.flush();
   RequestDispatcher rd =  getServletContext().getRequestDispatcher
              ("/servlet/itso.servjsp.servletapi.DispatcherInclude");
   rd.include(req, res);
   out.println("<P>End of request</BODY></HTML>");
}
}
```

Figure 67. Request dispatching servlet: calling servlet through include method

Note: You must flush the output before including the servlet, otherwise the output may be in the wrong order.

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DispatcherInclude extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   PrintWriter out = res.getWriter();
   out.println("<HR>I see you are a " + req.getParameter("title"));
   out.println("<P>End of include<HR>");
}
}
```

Figure 68. Request dispatching servlet: called servlet through include method

Resource usage

In request dispatching, discussed above, we can only dispatch to another active application resource, such as another servlet. We described passive application resources as elements such as HTML files.

We have shown that we can redirect to an application's resources, such as HTML files. But how do we get access to this application's resources directly, without having to redirect?

We can interact with an application's passive resources through the servlet context. The *ServletContext* object allows us access to these resources through the *getResource* or *getResourceAsStream* methods:

❑ *getResource:* This method returns a URL object to a resource known to the servlet context, and we can access the resource as a URL object:

```
URL url = servletContext.getResource("resourcefilename");
```

❑ *getResourceAsStream:* This method allows us to read the resources body directly as an InputStream that we can manipulate:

```
InputStream is = servletContext.getResourceAsStream("resourcefilename");
```

The *ResourceHandler* servlet (Figure 69) shows how we can implement these objects. In this example, we are accessing the HTML file shown in Figure 70. We reference it as a URL, through the *getResource* method, and also as an input stream, through the *getResourceAsStream* method.

Note that only the TITLE tag of the servlet is displayed in the output, not the title of the included HTML file.

```
package itso.servjsp.servletapi;

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ResourceHandler extends HTMLFormHandler {

public void service(HttpServletRequest req, HttpServletResponse res)
                                  throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>HTMLFormHandlerResource</TITLE></BODY>");
   out.println("<H2>Servlet API Example -
               HTMLFormHandlerResource</H2><HR>");
   ServletContext sc = getServletContext();
   URL url = sc.getResource("HTMLFormHandlerRedirect.html");
   out.println("URL name: " + url.getFile());
   out.println("<HR> Now the input file html:");
   BufferedReader in = new BufferedReader(new
     InputStreamReader(sc.getResourceAsStream("ResourceHandlerHTML.html")));
   String str;
   while ((str = in.readLine()) != null)
      out.println(str);
   in.close();
   out.println("<HR></BODY></HTML>");
}
}
```

Figure 69. Resource handler servlet: accessing passive application resources

```
<HTML>
<HEAD>
<TITLE>Servlet Examples - ResourceHandlerHTML</TITLE>
</HEAD>
<BODY>
<h4> Just a plain-old static HTML page </h4>
</BODY>
</HTML>
```

Figure 70. Resource handler html file: application resource

Sharing of objects in scope

We have seen that servlets may interact by calling each other's methods, redirecting, and dispatching (forwarding) their requests. Servlets may also communicate by accessing objects (attributes) which they have in common with other servlets. Attributes are available to servlets within the scope of request, session, and application. This section describes the ways in which objects may be shared at these different levels.

Request level scope

We have already seen how request level scope can be implemented, through the *RequestDispatcher forward* method. Here, the request object stays in scope as it is passed from resource to resource.

But in addition to the objects that are natively part of the request, how do we pass other objects along with the request? We do this by using the *request.setAttribute* method. This method allows us to store an object type by name, which we can later reference by that name as we forward the request to another servlet for processing. The object we store can be of a user-defined type, such as a JavaBean, which is accessible within the called servlet. Objects at this level remain in scope as long as the request is active, which is until the server sends back the response.

Figure 71 shows how we can set a request attribute called *count*. This code would be found in a calling (forwarding from) servlet in the request dispatching process.

```
//calledCount is the value we are storing to the attribute
int calledCount;
//Cast our int object to Integer, because cannot store native types
request.setAttribute("count", new Integer(calledCount));
```

Figure 71. Request attribute setting code snippet

Figure 72 shows how we can retrieve this attribute. This code would be found in a called (forwarded to) servlet in the request dispatching process.

```
//getting count attribute, and casting back to an int
Integer tempCount = (Integer)request.getAttribute("count");
int calledCount = tempCount.intValue();
out.println("Called count is: " + calledCount);
```

Figure 72. Request attribute getting code snippet

Session level scope

We have discussed the *HttpSession* object in “User sessions” on page 67. We showed how multiple requests from the same user to the same servlet can maintain state between request invocations. This information was managed at the user session level, and is referred to as session scope.

Session objects may be shared among other servlets and resources as well, within the scope of the same user. The only restriction is that the session object can only be shared among servlets that are within the same application server context of the original session; but this is true for all our servlet resources.

Session scope sharing example

In this example, we use the *UserSessionCounterSetter* (Figure 73) to generate and update the session counter with a variable each time that servlet is called by the user.

We also create another servlet, *UserSessionCounterGetter* (Figure 74), which gets the counter set in the *UserSessionCounterSetter* servlet. This demonstrates how these different servlets interact with the same *SaveServletStats* session object, through the *HttpSession* object.

We have also used a variable named *calledCount*, used in many examples thus far, to demonstrate the different values that are set when a counter is incremented by the servlet instance verses the user instance.

```
package itso.servjsp.servletapi
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserSessionCounterSetter extends HttpServlet {
   private int calledCount;

public void init(ServletConfig config) throws ServletException {
   super.init(config);
   calledCount = 0;
}
public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();

   HttpSession session = req.getSession(true);
   if (session.isNew() || session.getValue("usersession")==null) {
      session.putValue("usersession", new SaveServletStats());
   }
   SaveServletStats ustats =
                (SaveServletStats)session.getValue("usersession");
   calledCount++;
   ustats.calledCount++;
   out.println("<HTML><TITLE>SessionCounter</TITLE><BODY>");
   out.println("<H4>This servlet has been called: </H4><BR>");
   out.println("<B>" + calledCount + "</B> times since the servlet was loaded
             THIS servlet life-cycle session<BR>");
   out.println("<B>" + ustats.calledCount + "</B> times since the servlet was
             loaded by this user<BR>");
   out.println("</BODY></HTML>");
   out.close();
}
}
```

Figure 73. User session counter servlet: set user session data

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserSessionCounterGetter extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
   res.setContentType("TEXT/HTML");
   PrintWriter out = res.getWriter();
   HttpSession session = req.getSession(true);
   if (session.isNew() || session.getValue("usersession")==null) {
      session.putValue("usersession", new SaveServletStats());
   }
   SaveServletStats ustats =
        (SaveServletStats)session.getValue("usersession");
   out.println("<HTML><TITLE>SessionCounter</TITLE><BODY>");
   out.println("<H4>SessionCounter - UserSessionCounterGetter
               servlet</H4>");
   out.println("<B>" + ustats.calledCount + "</B> times since the
               UserSessionCounter servlet was loaded by this user<BR>");
   out.println("</BODY></HTML>");
}
}
```

Figure 74. User session counter servlet: get user session data

Application level scope

We introduced the term application context, where we described how Web applications are a grouping of servlets and resources that can share information and interact. This concept of Web applications is expressed in the *ServletContext* object.

The *ServletContext* object defines a servlet's view of the Web application within which the servlet is running, and gives the servlet the ability to access resources that are explicitly available to it. Using such an object, a servlet can log events, obtain URL references to resources, and get and set attributes at the application level that are available to other servlets in the same context, in much the same way as we set attributes at the request and session level.

The *ServletContext* is rooted at a specified path within a Web server. For example, a context could be located at `http://mywebserver/itsoservjsp`. All requests that start with the `/itsoservjsp` request path, which is known as the context path, will share the same servlet context.

ServletContext scope

Only one instance of a *ServletContext* may be available to the servlets of a Web application. Servlets that exist in WebSphere that are not part of a Web application are implicitly part of a default Web application called *default_app* in WebSphere Application Server.

ServletContext attributes

The Web application, through the *ServletContext* object, can also put object attributes into the context by name, using the *setAttribute* method. Any object in the context is available to other servlets that are part of the Web application through the *getAttribute* method. These objects may be of any object type, even JavaBeans.

ServletContext example

By being able to use the *ServletContext* object, we can share data among servlets at the application level, within the same servlet context.

In the next example, we demonstrate this servlet interaction technique. We have two simple servlets, *ContextSetAttribute* (Figure 75) that sets an attribute in the *ServletContext*, and *ContextGetAttribute (*Figure 76) that gets the attribute of the servlet context. These two servlets do not interact directly, but by the sharing of data in the *ServletContext* object.

Every time the *ContextSetAttribute* servlet is invoked, it increments a counter, and stores it in the servlet context, which is available to all servlets in the application. When *ContextGetAttribute* is called, it retrieves the last attribute value set, and prints it to the response.

We call this kind of scope application level scope. For the objects which we store in the session context, we can say they have application level scope. This concept ties directly into the application level scope of a JavaBean as discussed in Chapter 5, “JavaServer Pages” on page 95.

Servlets loaded by System class loader cannot be used for inter-servlet communication because they are not recognized in the servlet context, even though they may be part of the same application. To get a list of all other servlets in the servlet context, you can use the *getServletNames* method.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ContextSetAttribute extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                    throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SetContextAttribute</TITLE><BODY>");
   out.println("<H4>ServletContext example - Attribute Setter</H4><HR>");
   ServletContext sc = getServletContext();
   int calledCount = 0;
   if (sc.getAttribute("calledCount") != null) {
      Integer tempCount = (Integer)sc.getAttribute("calledCount");
      calledCount = tempCount.intValue();
   }
   out.println("ServletContext, server info: " + sc.getServerInfo() +
               "<BR>");
   out.println("SerlvetContext, real path: " + sc.getRealPath("") + "<BR>");
   out.println("The attribute 'calledCount' value we retrieved: " +
                calledCount + "<BR>");
   calledCount++;
   sc.setAttribute("calledCount", new Integer(calledCount));
   out.println("We set the ServletContext attribute 'calledCount' to: " +
                sc.getAttribute("calledCount"));
   out.println("</BODY><HTML>");
}
}
```

Figure 75. Context set attribute servlet: setting application scope attribute

```
package itso.servjsp.servletapi;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ContextGetAttribute extends HttpServlet {

protected void service(HttpServletRequest req, HttpServletResponse res)
                                   throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SetContextAttribute</TITLE><BODY>");
   out.println("<H4>ServletContext example - Attribute Getter</H4><HR>");
   ServletContext sc = getServletContext();
   int calledCount = 0;
   if (sc.getAttribute("calledCount") != null) {
      Integer tempCount = (Integer)sc.getAttribute("calledCount");
      calledCount = tempCount.intValue();
   }
   out.println("The attribute 'calledCount' value we retrieved: " +
              calledCount + "<BR>");
   out.println("</BODY><HTML>");
   out.close();
}
}
```

Figure 76. Context get attribute servlet: getting application scope attribute

New features of Java Servlet API 2.2

The following is a summary of new features available in the Java Servlet API 2.2. We did not use the API 2.2 convention in this chapter, because the WebSphere Application Server currently supports only Version 2.1.

Changes made to the specification

- ❑Introduction of the Web application concept
- ❑Introduction of the Web application archive files
- ❑Introduction of response buffering
- ❑Introduction of distributable servlets
- ❑Ability to get a RequestDispatcher by name
- ❑Ability to get a RequestDispatcher using a relative path

❑Internationalization improvements

❑Many clarifications of distributed servlet engine semantics

Changes made to the API

❑Added the *getServletName* method to the ServletConfig interface to allow a servlet to obtain the name by which it is known to the system, if any.

❑Added the *getInitParameter* and *getInitParameterNames* method to the ServletContext interface so that initialization parameters can be set at the application level to be shared by all servlets that are part of that application.

❑Added the *getLocale* method to the ServletRequest interface to aid in determining what locale the client is in.

❑Added the *isSecure* method to the ServletRequest interface to indicate whether or not the request was transmitted via a secure transport such as HTTPS.

❑Replaced the construction methods of *UnavailableException*, as existing constructor signatures caused some amount of developer confusion. These constructors have been replaced by simpler signatures.

❑Added the *getHeaders* method to the HttpServletRequest interface to allow all the headers associated with a particular name to be retrieved from the request.

❑Added the *getContextPath* method to the HttpServletRequest interface so that the part of the request path associated with a Web application can be obtained.

❑Added the *isUserInRole* and *getUserPrinciple* methods to the HttpServletRequest method to allow servlets to use an abstract role based authentication.

❑Added the *addHeader*, *addIntHeader*, and *addDateHeader* methods to the HttpServletResponse interface to allow multiple headers to be created with the same header name.

❑Added the *getAttribute*, *getAttributeNames*, *setAttribute*, and *removeAttribute* methods to the HttpSession interface to improve the naming conventions of the API. The *getValue*, *getValueNames*, *setValue*, and *removeValue* methods are deprecated as part of this change.

Summary

We have covered a lot of ground in this chapter:

- ❑We provided an overview of the Java Servlet API, and described how servlets interact with the Web application server.
- ❑We described the servlet life-cycle, and discussed the important servlet life-cycle methods of init, service, doGet, doPost, and destroy.
- ❑We introduced a number of servlet examples. In these examples, we covered the topics of:
  - Life-cycle execution
  - Persistence
  - Multi-threading
  - Servlet initialization
  - State maintaining mechanisms, including:
    - Cookies
    - URL rewriting
    - HttpSession objects
- ❑We discussed various servlet interaction techniques, including:
  - Servlet collaboration
  - Calling servlets from servlets
  - Response redirection
  - Request dispatching
  - Resource usage
  - Sharing of object in scope

5 JavaServer Pages

In this chapter, we discuss JavaServer Pages and how to use the JavaServer Pages Version 1.0 specification to create dynamic Web pages.

This chapter begins with a high-level introduction to JavaServer Pages, how they interact with the application server, and what benefits the technology offers over previous technologies. Later in the chapter, we provide details on JavaServer Page syntax elements and offer simple working examples which demonstrate the use of these elements. Finally, we describe some of the differences between the JSP 0.91 specification and the 1.0 specification.

In addition to reading this chapter, you may also want to refer to Appendix A, “JSP tag syntax” on page 401 for a summary of JSP tags and syntax.

If you are unfamiliar with Java servlets, we suggest you read “Servlets” on page 41 prior to reading this chapter.

If you want to run the examples presented here, refer to Chapter 6, “WebSphere Application Server” on page 123 and to Chapter 7, “Development and testing with VisualAge for Java” on page 167. All the examples are provided on the Internet (see Appendix C, “Using the additional material” on page 417).

Overview

JavaServer Pages (JSPs) are similar to HTML files, but provide the ability to display dynamic content within Web pages. JSP technology was developed by Sun Microsystems to separate the development of dynamic Web page content from static HTML page design. The result of this separation means that the page design can change without the need to alter the underlying dynamic content of the page. This is useful in the development life-cycle because the Web page designers do not have to know how to create the dynamic content, but simply have to know where to place the dynamic content within the page.

To facilitate embedding of dynamic content, JSPs use a number of *tags* that enable the page designer to insert the properties of a JavaBean object and script elements into a JSP file. A number of development tools, such as the WebSphere Studio Page Designer, can be used to visually create a page containing dynamic contents based on the properties of Java beans (this is covered in more detail in Chapter 8, “Development with WebSphere Studio” on page 227).

Here are some of the advantages of using JSP technology over other methods of dynamic content creation:

❑Separation of dynamic and static content

This allows for the separation of application logic and Web page design, reducing the complexity of Web site development and making the site easier to maintain.

❑Platform independence

Because JSP technology is Java-based, it is platform independent. JSPs can run on any nearly any Web application server. JSPs can be developed on any platform and viewed by any browser because the output of a compiled JSP page is HTML.

❑Component reuse

Using JavaBeans and Enterprise JavaBeans, JSPs leverage the inherent reusability offered by these technologies. This enables developers to share components with other developers or their client community, which can speed up Web site development.

❑Scripting and tags

JSPs support both embedded JavaScript and tags. JavaScript is typically used to add page-level functionality to the JSP. Tags provide an easy way to embed and modify JavaBean properties and to specify other directives and actions.

Throughout this book, we use the JSP 1.0 specification; however, WebSphere Application Server 3.02 and Visual Age for Java 3.02 support both JSP 1.0 and JSP 0.91 specifications.

At the time of writing, the JSP 1.1 Specification - Final Release was available at http://java.sun.com/products/jsp/.

How JavaServer Pages work

JavaServer Pages are made operable by having their contents (HTML tags, JSP tags and scripts) translated into a servlet by the application server. This process is responsible for translating both the dynamic and static elements declared within the JSP file into Java servlet code that delivers the translated contents through the Web server output stream to the browser.

Because JSPs are server-side technology, the processing of both the static and dynamic elements of the page occurs in the server. The architecture of a JSP/servlet-enabled Web site is often referred to as *thin-client* because most of the business logic is executed on the server.

The following process outlines the tasks performed on a JSP file on the *first invocation* of the file or when the underlying JSP file is changed by the developer (Figure 77):

- ❑The Web browser makes a request to the JSP page.
- ❑The JSP engine parses the contents of the JSP file.
- ❑The JSP engine creates temporary servlet source code based on the contents of the JSP. The generated servlet is responsible for rendering the static elements of the JSP specified at design time in addition to creating the dynamic elements of the page.
- ❑The servlet source code is compiled by the Java compiler into a servlet class file.
- ❑The servlet is instantiated. The *init* and *service* methods of the servlet are called, and the servlet logic is executed.
- ❑The combination of static HTML and graphics combined with the dynamic elements specified in the original JSP page definition are sent to the Web browser through the output stream of the servlet's response object.

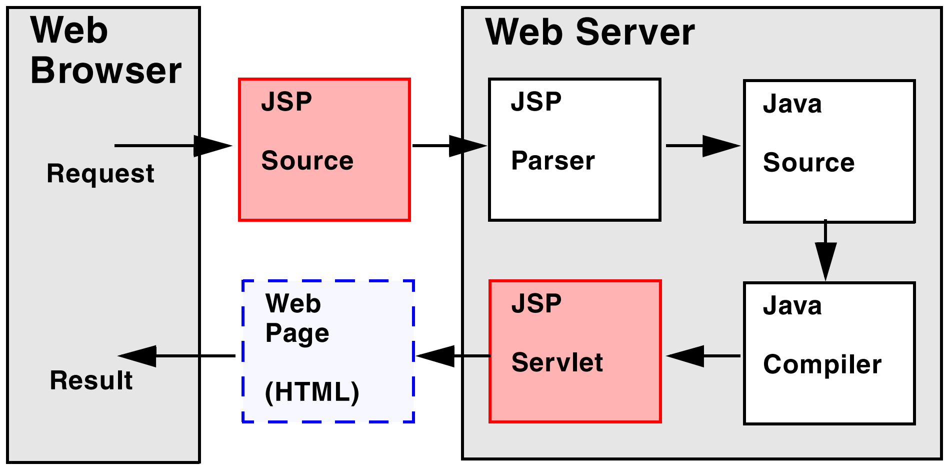


Figure 77. The JSP processing life-cycle on first-time invocation

Subsequent invocations of the JSP file will simply invoke the *service* method of the servlet created by the above process to serve the content to the Web browser. The servlet produced as a result of the above process remains in service until the application server is stopped, the servlet is manually unloaded, or a change is made to the underlying file, causing recompilation.

In the source code examples provided with this book, we have included the compiled JSP source for the *DateDisplay.jsp* (Figure 79 on page 108) in the file *_DateDisplay_xjsp.java*. This code is useful in understanding the relationship between JSPs and servlets and to help you understand the role of the JSP engine in converting a JSP to a servlet.

Components of JavaServer Pages

JavaServer Pages are composed of standard HTML tags and JSP tags. The available JSP tags defined in the JSP 1.0 specification are categorized as follows:

❑Directives
❑Declarations
❑Scriptlets
❑Comments
❑Expressions

This section describes each of these categories in more detail.

HTML tags

JavaServer Pages support all HTML tags. For a listing of HTML tags, refer to your HTML manual.

JSP directives

A JSP directive is a global definition sent to the JSP engine that remains valid regardless of any specific requests made to the JSP page. A directive always appears at the top of the JSP file, before any other JSP tags. This is due to the way the JSP parsing engine produces servlet code from the JSP file.

The syntax of a directive is:

```
<%@ directive directive_attr_name = value %>
```

Directives are grouped as follows:

page

The *page* directive defines page dependent attributes to the JSP engine.

```
<%@ page language="java" buffer="none" isThreadSafe="yes"
        errorPage="/error.jsp" %>
```

The attributes of the page directive are listed in Table 1.

Table 1. Attributes of the page directive

Attribute Name	Description
language	Identifies the scripting language used in scriptlets in the JSP file or any of its included files. JSP supports only the value of "java". WebSphere extensions provide support for other scripting languages. `<%@ page language = "java" %>`
extends	The fully-qualified name of the superclass for which this JSP page will be derived. Using this attribute can effect the JSP engine's ability to select specialized superclasses based on the JSP file content, and should be used with care.
import	When the language attribute of "java" is defined, the import attribute specifies the additional files containing the types used within the scripting environment. `<%@ page import = "java.util.*" %>`

Attribute Name	Description
session "true" | "false"	If *true*, specifies that the page will participate in an HTTP session and enables the JSP file access to the implicit session object. The default value is *true*.
buffer "none" | "sizekb"	Indicates the buffer size for the JspWriter. If *none*, the output from the JSP is written directly to the ServletResponse PrintWriter object. Any other value results in the JspWriter buffering the output up to the specified size. The buffer is flushed in accordance with the value of the autoFlush attribute. The default buffer size is no less than 8kb.
autoFlush "true" | "false"	If *true*, the buffer will be flushed automatically. If false, an exception is raised when the buffer becomes full. The default value is *true*.
isThreadSafe "true" | "false"	If *true*, the JSP processor may send multiple outstanding client requests to the page concurrently. If *false*, the JSP processor sends outstanding client requests to the page consecutively, in the same order in which they were received. The default is *true*.
info	Allows the definition of a string value that can be retrieved using Servlet.getServletInfo().
errorPage	Specifies the URL to be directed to for error handling if an exception is thrown and not caught within the page implementation. In the JSP 1.0 specification, this URL must point to a JSP page.
isErrorPage "true" | "false"	Identifies that the JSP page refers to a URL identified in another JSP's errorPage attribute. When this value is *true*, the implicit variable *exception* is defined, and its value set to reference the Throwable object of the JSP source file which causes the error.
contentType	Specifies the character encoding and MIME type of the JSP response. Default value for `contentType` is *text/html*. Default value for `charSet` is *ISO-8859-1*. The syntax format is: `contentType="text/html; charSet=ISO-8859-1"`

include

The *include* directive allows substitution of text or code to occur at translation time. You can use the include directive to provide a standard header on each JSP page, for example:

```
<%@ include file="copyright.html" %>
```

The include directive has the attributes shown in Table 2.

Table 2. Attributes for the include directive

Attribute Name	Description
file	Directs the JSP engine to substitute the text or code specified by file or URL reference. The URL reference can be another JSP file.

taglib

The *taglib* directive allows custom extensions to be made to the tags known to the JSP engine. This tag is an advanced feature. Refer to the Sun JavaServer Page 1.0 specification for more information about this tag.

Declarations

A declaration block contains Java variables and methods that are called from an *expression* block within the JSP file. Code within a declaration block is usually written in Java, however, the WebSphere application server supports declaration blocks containing other script syntax. Code within a declaration block is often used to perform additional processing on the dynamic data generated by a JavaBean property.

The syntax of a declaration is:

```
<%! declaration(s) %>
```

For example:

```
<%!
   private int getDateCount = 0;
   private String getDate(GregorianCalendar gc1)
      { ...method body here...}
%>
```

Scriptlets

JSP supports embedding of Java code fragments within a JSP by using a *scriptlet* block. Scriptlets are used to embed small code blocks within the JSP page, rather than to declare entire methods as performed in a declarations block. The syntax for a scriptlet is:

```
<% scriptlet %>
```

The following example uses a scriptlet to output an HTML message based on the time of day. You can see that the HTML elements appear *outside* the script declarations.

```
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
   {%>
      How are you this morning ?
<% } else
   { %>
      How are you this afternoon ?
<% } %>
```

Comments

You can use two types of comments within a JSP. The first comment style, known as an *output comment*, enables the comment to appear in the output stream on the browser. This comment is an HTML formatted comment whose syntax is:

```
<!-- comments ... -->
```

The second comment style is used to fully exclude the commented block from the output and is commonly used when uncommenting a block of code that so that the commented block is never delivered to the browser. The syntax is:

```
<%-- comment text --%>
```

You can also create comments containing dynamic content by embedding a scriptlet tag inside a comment tag. For example:

```
<!-- comment text <%= expression %> more comment text ->
```

Expressions

Expressions are scriptlet fragments whose results can be converted to String objects and subsequently fed to the output stream for display in a browser. The syntax for an expression is:

```
<%= expression %>
```

Typically, expressions are used to execute and display the String representation of variables and methods declared within the declarations section of the JSP, or from JavaBeans that are accessed by the JSP. If the conversion of the expression result is unsuccessful, a *ClassCastException* is thrown at the time of the request.

The following example calls the *incrementCounter* method declared in the declarations block and prints the result.

```
<%= incrementCounter() %>
```

All primitive types such as short, int, and long can be automatically converted to Strings. Your own classes must provide a *toString* method for String conversion.

WebSphere extensions to JSP scripting

WebSphere Application Server Version 3 offers a number of enhancements over the JSP 1.0 specification and includes the ability to:

- ❑Use non-Java scripting languages within JSP pages.
- ❑Use multiple scripting languages within the same JSP file.

You can use any of the Bean Scripting Framework (BSF) 1.0 compliant languages in your JSP by specifying it within the *language_name* attribute of the `page` directive. Information about using BSF compliant scripting languages can be found at

http://www.alphaWorks.ibm.com/tech/bsf

See Table 3 for semantics of using multiple scripting languages within a JSP.

Table 3. WebSphere scripting language extensions

SpecifySyntax
<jsp:scriptlet language="*language_name*">
<jsp:expr language="*language_name*">
<jsp:declaration language="*language_name*">

Accessing implicit objects

When you are writing scriptlets or expressions, there are a number of objects that you have automatic access to as part of the JSP standard without having to fully declare them or import them. Table 4 summarizes these implicit objects available in JSP 1.0.

You can use these implicit objects directly in your code. The following code snippet is an example of accessing the `out` implicit object to display a line of text in the browser:

```
out.println("Here is the <b>Date Display JSP</b>");
```

Table 4. Summary of implicitly declared objects

Object name	Type	Description
request	javax.servlet.HttpServletRequest	The request triggering the service invocation
response	javax.servlet.HttpServletResponse	The response to the request
pageContext	javax.servlet.jsp.PageContext	Page context of this JSP. By accessing this object, you have access to a number of convenience objects and methods such as getException, getPage, and getSession providing an explicit method of accessing JSP implementation-specific objects.
session	javax.servlet.http.HttpSession	Session object created for the requesting client
application	javax.servlet.ServletContext	The servlet context as obtained from the servlet configuration object
out	javax.servlet.jsp.JspWriter	Output stream writer
config	javax.servlet.ServletConfig	ServletConfig for this JSP
page	java.lang.Object	Instance of this page's implementation class processing the current request

Putting it all together

Figure 78 shows an example which combines many of the JSP components previously discussed in this chapter. In the example, the current date is displayed together with a count of the number of times the *getDate* function has been called.

Note that the counter continues to increment until the servlet is manually stopped, the Application Server is restarted, or the JSP page is modified, forcing a page compilation to occur.

Note also that the *calledCount* variable is a private variable declared *outside* of the getDate function. This has implications in a multi-user environment, as each browser accessing the servlet causes the count to be updated, and is therefore not thread-safe. For thread-safety, the variable should be accessed in an access function that implements the *synchronized* modifier.

```
<html>
<title>Date Display</title>
<body>

<!-- D I R E C T I V E S -->
<%@ page language = "java" %>
<%@ page import = "java.util.*" %>
<%@ page contentType = "TEXT/HTML" %>

<!-- S C R I P T L E T S-->
<H3>
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
    {%>
    How are you this morning,
    <%} else {%>
    How are you this afternoon,
    <% }
%>
WebSphere 3 User ?
</H3>
<HR>

<!-- A C C E S S I N G   I M P L I C I T   O B J E C T S -->
<% out.println("Here is the <b>Date Display JSP</b>"); %>

<!-- D E C L A R A T I O N S -->

<%!

private int calledCount = 0;
private String getDate(GregorianCalendar gcalendar) {

    StringBuffer dateStr = new StringBuffer();
    dateStr.append(gcalendar.get(Calendar.DATE));
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.MONTH) + 1);
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.YEAR));
    return (dateStr.toString());
}

private int incrementCounter() {
    return (++calledCount);
}

%>

<H1> Today's Date is: <%= getDate(new GregorianCalendar()) %> </H1>
<H1> This page has been called: <%= incrementCounter() %> time(s)</H1>
</body>
</html>
```

Figure 78. Sample JSP demonstrating JSP components (DateDisplay.jsp)

JSP interactions

There are a number of methods that a JSP can use to interact with the Web environment. Primarily, a JSP will use a JavaBean object to present dynamic content. However, a JSP can also invoke another JSP page by URL, by including another JSP or HTML page in the *include* directive, or by calling a servlet.

This section describes these interactions.

Invoking a JSP by URL

A JSP can be invoked by URL, from within the *<FORM>* tag of a JSP or HTML page, or from another JSP.

To invoke a JSP by URL, use the syntax:

```
http://servername/path/filename.jsp
```

For example, to invoke the DateDisplay.jsp, use this URL:

```
http://localhost/itsoservjsp/DateDisplay.jsp              <== WebSphere
http://localhost:8080/itsoservjsp/DateDisplay.jsp         <== VA Java
```

Calling a servlet from a JSP

You can invoke a servlet from a JSP either as an action on a form, or directly through the *jsp:include* or *jsp:forward* tags.

Form action

Typically, you want to call a servlet as a result of an action performed on a JavaServer Page. For example, you may want to process some data entered by the user in an HTML form when they click on the *Submit* button.

To invoke a servlet within the HTML <FORM> tag, the syntax is:

```
<FORM METHOD="POST|GET" ACTION="application_URI/JSP_URL">
    <!-- Other tags such as text boxes and buttons go here -->
</FORM>
```

For example:

```
<form method="POST"
      action="/itsoservjsp/servlet/itso.servjsp.jspsamples.DateDisplayServlet">
```

Figure 79 shows the code to call the *DateDisplayServlet* from within a JSP.

```
<HTML>
<HEAD> <TITLE> Call Servlet from JSP </TITLE> </HEAD>
<CENTER>
<H1> Call Servlet from JSP </H1>
<FORM method="POST"
  action="/itsoservjsp/servlet/itso.servjsp.jspsamples.DateDisplayServlet">

   <H2> DateDisplay Servlet Launcher </H2>
   Click the button below to display the current date
   <P> <INPUT type="submit" name="CALL_SERVLET" value="Call the Servlet">
</FORM>
</CENTER>
</BODY></HTML>
```

Figure 79. Sample JSP invoking a servlet from a form (JspToServlet.jsp)

JSP include tag

You can include the output of a servlet in a JSP using the jsp.include tag:

```
<jsp:include page="/servlet/itso.servjsp.servletapi.SHTMLServlet" />
```

Figure 80 shows a JSP that includes the servlet that we used in Figure 57 on page 73.

```
<HTML><BODY>
<H2> JSP to Servlet </H2>
<HR>
<jsp:include page="/servlet/itso.servjsp.servletapi.SHTMLServlet" />
<HR>
<H2>End of servlet include</H2>
</HTML></BODY>
```

Figure 80. Sample JSP including a servlet (JspInclude.jsp)

When you run this JSP the output of the servlet is imbedded in the JSP output.

JSP forward tag

You can forward processing from a JSP to a servlet using the jsp.forward tag:

```
<jsp:forward page="/servlet/itso.servjsp.servletapi.SHTMLServlet" />
```

Figure 81 shows a JSP that forwards processing to the servlet that we used in Figure 57 on page 73.

```
<HTML><BODY>
<H2> JSP to Servlet </H2>
<HR>
<jsp:forward page="/servlet/itso.servjsp.servletapi.SHTMLServlet" />
<HR>
<H2>End of servlet include</H2>
</HTML></BODY>
```

Figure 81. Sample JSP forwarding processing to a servlet (JspForward.jsp)

When you run this JSP, the output of the processing servlet replaces the output of the JSP. All output of the JSP is lost.

Calling a JSP from a servlet

Figure 82 shows the DateDisplayServlet's *doPost* method, which is called when the *Submit* button is clicked. The servlet simply calls the *sendRedirect* method of the HttpServletResponse object, directing the response to the *DateDisplay.jsp*. This example simply demonstrates the redirection capability of the response object. In reality, the doPost method could invoke other methods which process the form data, instantiate other beans that perform the business logic, and finally redirect the user to the JSP.

```
import javax.servlet.http.*;

public class DateDisplayServlet extends HttpServlet {

public void doPost(HttpServletRequest req, HttpServletResponse resp)
             throws javax.servlet.ServletException, java.io.IOException {
   // Redirect to the DateDisplay JSP page
   resp.sendRedirect("/DateDisplay.jsp");

   // alternate call to JSP
   // getServletContext().getRequestDispatcher("/DateDisplay.jsp").forward(req,resp);
}
```

Figure 82. DateDisplayServlet demonstrating simple redirection

You can also use the RequestDispatcher object (see “Request dispatching” on page 81) to invoke a JSP:

```
getServletContext().getRequestDispatcher("/DateDisplay.jsp").forward(req,resp);
```

PageListServlet Class

IBM provides the *PageListServlet* class in the *com.ibm.servlet* package. This is a subclass of HttpServlet that provides a *callPage* method to invoke JSPs.

Servlets generated by the WebSphere Studio wizards are subclasses of the PageListServlet class. Such a servlet must have an associated servlet configuration file (.servlet) that specifies all the possible JSPs that the servlet may invoke. See “Servlet configuration file” on page 59 for a description.

A typical call to invoke a JSP from a PageListServlet is:

```
callPage("myJSP", request, response);
```

The name of the JSP can be a short name (alias) that is assigned to the real file name of the JSP in the servlet configuration file (Figure 83).

```
<?xml version="1.0"?>
<!-- This file was generated by IBM WebSphere Studio 3.0.2 -->
<servlet>
    <page-list>
      <default-page>
        <uri>/itsoservjsp/photo/photoResults.jsp</uri>
      </default-page>
      <error-page>
        <uri>/itsoservjsp/photo/photoError.jsp</uri>
      </error-page>
      <page>
        <uri>/itsoservjsp/photo/photoNoData.jsp</uri>
        <page-name>com.ibm.webtools.runtime.NoDataException</page-name>
      </page>
      <page>
        <uri>/itsoservjsp/photo/photoSpecial.jsp</uri>
        <page-name>myJSP</page-name>
      </page>
    </page-list>
    <code>itso.servjsp.photo.photo</code>
</servlet>
```

Figure 83. Servlet configuration file with JSP names

Because the JSP names used in the callPage method of the servlet are aliases, a change of directory can be accomplished by changing the servlet configuration file, without touching the servlet code.

Invoking a JSP from a JSP

To invoke a JSP file from another JSP file, you can:

- ❑Specify the URL of the second JSP file on the FORM ACTION attribute:

  ```
  <FORM action="/itsoservjsp/DateDisplay.jsp">
  ```

- ❑Specify the URL of the second JSP file in an anchor tag HREF attribute:

  ```
  <a href="JSP_URL"> reference-text </a>
  ```

- ❑Use the javax.servlet.http.RequestDispatcher.forward method to invoke the second JSP file (see “Request dispatching” on page 81). This is the same as using the *jsp:forward* tag.

Creating dynamic content in JSPs

This section discusses some of the more commonly used tags available in the JSP 1.0 specification which assist you in creating dynamic content within a JSP. In addition to describing some of the more commonly used JSP tags, we also describe how to use the WebSphere-specific tags that provide support for relational database access.

For a complete description of all tags supported by the JSP 1.0 specification, please refer to the Sun JavaServer Pages Specification Version 1.0 available on the Sun Web site.

Standard JSP tags

jsp:useBean

The *jsp:useBean* tag is used to declare a JavaBean object that you want to use within the JSP. Before you can use the *jsp:getProperty* and *jsp:setProperty* tags, you must have first declared your JavaBean using the jsp:useBean tag. When the jsp:useBean tag is processed, the application server performs a lookup of the specified given Java object using the values specified in the *id* and *scope* attributes. If the object is not found, it will attempt to create it using the values specified in the *scope* and *class* attributes.

The syntax for inserting a JavaBean is:

```
<jsp:useBean id="beanInstanceName" scope="page|request|session|application"
             typespec>
   optional scriptlets and tags
</jsp:useBean>
```

Here, *typespec* can be declared using any of the following variations:

```
class="package.class"
type="package.class"
type="package.class" beanName="package.class"
```

You can also embed scriptlets and tags such as *jsp:getProperty* within the jsp:useBean declaration which will be executed upon creation of the bean. This is often used to modify properties of a bean immediately after it has been created.

An example of a simple form of bean instantiation is:

```
<jsp:useBean id ="DateDisplayBean"
             class="itso.servjsp.jspsamples.DateDisplayBean"/>
```

This example tries to locate an instance of the *DateDisplayBean* class. If no instance exists, a new instance is created. The instance can then be accessed within the JSP using the specified *id* of DateDisplayBean.

Table 5 describes the jsp:useBean attributes.

Table 5. jsp:useBean attributes

Parameter Name	Description
id	Identifies the object name within the name space of the specified scope. This name is used to reference the bean throughout the JSP file and is case sensitive.
scope	Valid values are page, request, session, and application. If omitted, the value defaults to *page* scope. ***page***: Objects declared with page scope are only valid until the response is sent back from the server or until the request is forwarded elsewhere. References to objects in page scope are only valid within the page where the object is declared. Objects declared in page scope are stored in the `pagecontext` object. ***request***: Objects declared within `request` scope are valid for the duration of the request and are accessible if the request is forwarded to a resource in the same runtime. Objects referenced in request scope are stored in the `request` object. ***session***: Session-scope objects are available for the duration of the session provided that the page is made “session aware” using the `page` directive. ***application***: Application-scope objects are available from pages that are processing requests within the same Web application (as defined in the application server setup) and are valid until the `ServletContext` object is reclaimed by the application server. Objects with this scope are stored in the `application` object.

Parameter Name	Description
class	The name of the object's implementation class, for example: itso.servjsp.jspsamples.DateDisplayBean. This value is case sensitive. Specify the class attribute if you want to instantiate the bean if it does not already exist within the specified scope.
beanName	Specifies the class name or serialized file (.ser) containing the bean which is used when first creating the bean.
type	Identifies the type of the specified object. This allows the scripting variables type to be declared as the class itself, the superclass or an interface implemented by the class. By specifying the type attribute, you can avoid automatic instantiation of the bean if it does not already exist within the specified scope, effectively reproducing the behavior of the JSP .91 `create="yes/no"` attribute on the <BEAN> tag. The default value is the value specified in the *class* attribute. If the object is not of the specified type, you may receive a `java.lang.ClassCastException`.

If you do *not* want to automatically instantiate a bean if it does not already exist within the specified scope, use the *type* attribute rather than the *beanName* or *class* attributes. The following line will result in an InstantiationException if the object specified by the `type` attribute does not exist in the session scope, and as a result, the bean will not be instantiated.

```
<jsp:useBean id="DateDisplayBean"
          type="itso.servjsp.jspsamples.DateDisplayBean" scope="session"/>
```

jsp:getProperty

Once the bean has been declared with jsp:useBean, you can access its exposed properties through the *jsp:getProperty* tag, which inserts the String value of the primitive type or object into the output stream. For primitive types, the conversion to String is performed automatically. For object types, the *toString* method of the object is called.

The syntax for the jsp:getProperty tag is

```
<jsp:getProperty name="beanName" property="propertyName"/>
```

The jsp:getProperty tag has a number of attributes as defined in Table 6.

Table 6. jsp:getProperty attributes

Attribute Name	Description
name	The name (id) of the bean instance specified in the jsp:useBean tag.
property	The name of the property to get.

Figure 84 shows the source code of a JavaBean called *DateDisplayBean* that we are referencing in a JSP.

```
package itso.servjsp.jspsamples;
import java.util.*;

public class DateDisplayBean {
   private int counter = 0;
   private String dateString = null;

public DateDisplayBean() {
   super();
   dateString = buildDateString(new GregorianCalendar());
   counter = 0;
}
public String buildDateString(GregorianCalendar gcalendar) {
   StringBuffer dateStr = new StringBuffer();
   dateStr.append(gcalendar.get(Calendar.DATE));
   dateStr.append("/");
   dateStr.append(gcalendar.get(Calendar.MONTH) + 1);
   dateStr.append("/");
   dateStr.append(gcalendar.get(Calendar.YEAR));
   return dateStr.toString();
}
public int getCounter() {
   return counter;
}
public void setCounter(int newCounter) {
   counter = newCounter;
}
public java.lang.String getDateString() {
   counter++;
   return dateString;
}
}
```

Figure 84. JavaBean to be used by a JSP (DateDisplayBean.java)

The example JSP file in Figure 85 below declares the DateDisplayBean and displays the two properties, *dateString* and *counter*.

```
<html> <title>Date Display Bean </title> <body>
<H1> Date Display with JSP and JavaBean </H1>
<jsp:useBean id="DateDisplayBean"
         class="itso.servjsp.jspsamples.DateDisplayBean" scope="session" />
<H2> Today's Date is:
   <jsp:getProperty name="DateDisplayBean" property="dateString"/>
</H2>
<H2> This page has been called:
   <jsp:getProperty name="DateDisplayBean" property="counter"/>
   time(s)
</H2>
</body> </html>
```

Figure 85. JSP with jsp:useBean and jsp:getProperty (JspWithBean.jsp)

jsp:setProperty

The properties of beans can be set by using the *jsp:setProperty* tag. The syntax for this tag is:

```
<jsp:setProperty name="beanName" prop_expr/>
```

For example, to initialize the counter variable used in Figure 85, you could use the code:

```
<jsp:setProperty name="DateDisplayBean" property="counter" value="0"/>
```

The jsp:setProperty tag has a number of attributes, as defined in Table 7.

Table 7. jsp:setProperty attibutes

Attribute Name	Description
name	The name (id) of the bean instance specified in the jsp:useBean tag.
property	The name of the property to set. By setting this value to “*“, you can automate the setting of properties, provided that form-element names match the property name. For example, if a bean has a property called *dateString*, and the JSP page contains a text box named *dateString*, then the dateString property of the bean will be looked up and set automatically. For this feature to work, your beans must conform to the JavaBeans API specification 1.0.

Attribute Name	Description
param	The request parameter name to give to the Bean property. Request parameters usually refer to the names of HTML form elements, and are used to implicitly set the value of a particular bean property based on the value of the HTML form element. This attribute cannot be used with the `value` attribute.
value	The new value for the property.

WebSphere-specific tags

WebSphere provides a number of extensions to the JSP language.

tsx:dbconnect

The *tsx:dbconnect* tag is required to connect to JDBC or ODBC databases. This tag does not actually make the connection, but rather sets the connection attributes used by the *tsx:dbquery* and *tsx:dbmodify* tags, which are responsible for making the database connection before interacting with the database.

The syntax of the tsx:dbconnect tag is:

```
<tsx:dbconnect id="connection_id"
   userid="db_user" passwd="db_password"
   url="jdbc:subprotocol:database"
   driver="database_driver_name" >
</tsx:dbconnect>
```

The *tsx:dbconnect* tag has the attributes shown in Table 8.

Table 8. tsx:dbconnect attributes

Attribute Name	Description
id	The name of the connection. This tag is used by the tsx:dbquery and tsx:dbmodify tags as a reference to the connection.
userid	A valid database user ID. If omitted, the user ID and password should be specified using the *tsx:userid* tag.
password	The password for the database.If omitted, the user ID and password should be specified using the *tsx:password* tag.
url	The JDBC URL of the database, for example: `url="jdbc:db2:sample"`

Attribute Name	Description
driver	The name of the driver used to establish the connection: `driver="COM.ibm.db2.jdbc.app.DB2Driver"`

The tsx:dbconnect tag does *not* support JNDI datasource lookup, as in the example:

```
url="jdbc/sample"
```

tsx:dbquery

The *tsx:dbquery* tag provides the mechanism to get a result set containing database data. It relies on the connection attributes specified by the tsx:dbconnect tag, which must be defined before this tag can be used.

The responsibilities of the tsx:dbquery tag are to:

❑Reference the connection object attributes created by tsx:dbconnect.

❑Establish the database connection.

❑Retrieve and cache the result set data.

❑Release the connection resource.

The tsx:dbquery tag has the following syntax:

```
<tsx:dbquery id="query_id" connection="connection_id" limit="value">
   SELECT statement ....
</tsx:dbquery>
```

The tsx:dbquery tag has the attributes shown in Table 9.

Table 9. tsx:dbquery attributes

Attribute Name	Description
id	The name of the query. This becomes the name of the result bean.
connection	The name given to the *id* attribute specified in the tsx:dbconnect tag.
limit	Specifies the maximum number of rows to return in the result set. This attribute is optional.

When a tsx:dbquery tag is compiled by the JSP engine, the name specified in the *id* parameter is used to create a JavaBean of that name containing the result set. The bean will also have properties that match the names of the database columns returned in the result set. Figure 87 on page 120 demonstrates using these properties to embed values from the record set within the formatted output.

If you want to customize the property names within the bean, you can use a column name alias in the SQL query. The SQL statement below will create a bean with a property of Dept rather than WORKDEPT:

```
Select WORKDEPT As Dept from Department
```

tsx:dbmodify

The *tsx:dbmodify* tag enables you to perform INSERT and UPDATE SQL commands on a database. Similar to the tsx:dbquery tag, it relies on the connection attributes specified by the tsx:dbconnect tag, which must be defined before this tag can be used.

The tsx:dbmodify tag has the following syntax:

```
<tsx:dbmodify connection="connection_id">
   INSERT/UPDATE/DELETE SQL statement ....
</tsx:dbmodify>
```

The attributes for the tsx:dbmodify tag are defined in Table 10.

Table 10. tsx:dbmodify attributes

Attribute Name	Description
connection	The name given to the *id* attribute specified in the tsx:dbconnect tag.

The example in Figure 86 demonstrates how to insert a row into the EMPLOYEE table.

```
<tsx:dbmodify connection="conn" >
   insert into EMPLOYEE
      (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, EDLEVEL)
   values (
      '<%= request.getParameter("EMPNO")    %>',
      '<%= request.getParameter("FIRSTNME") %>',
      '<%= request.getParameter("MIDINIT")  %>',
      '<%= request.getParameter("LASTNAME") %>',
      '<%= request.getParameter("WORKDEPT") %>',
      '<%= request.getParameter("EDLEVEL")  %>')
</tsx:dbmodify>
```

Figure 86. Using the tsx:dbmodify tag to insert a row in the sample database

tsx:getProperty

The *tsx:getProperty t*ag is a WebSphere extension to the jsp:getProperty tag. This implementation includes all the functionality of jsp:getProperty and adds the ability to introspect a database bean created by the tsx:dbquery or tsx:dbmodify tags. You can use this tag to get properties from your own JavaBeans, or to get properties from the JavaBeans created by a call to tsx:dbquery where the beans properties refer to database columns, as in the following example:

```
<tsx:getProperty name="queryid" property="DEPTNAME"/>
```

tsx:repeat

The *tsx:repeat* tag is used to iterate over a database query result set using the optional *start* and *end* values as the bounding indexes for the iteration. The syntax is:

```
<tsx:repeat index=name start=start_index end=end_index>
</tsx:repeat>
```

The start and end attributes are optional attributes that can be implicitly or explicitly set. By default, these values are 0 and the upper bound of the result set respectively. You can use either attribute on its own or as a pair.

The iteration is complete when either the end value has been reached or an ArrayIndexOutOfBounds exception is thrown. No output is written until a complete iteration of the tsx.repeat block is complete. If an ArrayIndexOutOfBounds exception is thrown during an iteration, no output is written, and the repeat block is terminated.

The attributes for the tsx:repeat tag are listed in Table 11.

Table 11. tsx:repeat attributes

Attribute Name	Description
index	The name of the index. This name has JSP file scope and is case sensitive.
start	The optional start index for the iteration, with a default of 0.
end	The optional end index for the interaction. The maximum value for this attribute is 2,147,483,647. If the end attribute is less than the start attribute, it is ignored. The default is the number of available values in the bean.

Nesting of tsx:repeat tags is permissible and can be used to provide sub-category information during a query. For example, for each department in the department table, you might wish to list each employee associated with the department. You would do this by nesting tsx:repeat tags as shown in Figure 87.

```
<HTML>
<HEAD> <TITLE>Call Servlet</TITLE> </HEAD>

<H1> Department Listing with a JSP and TSX tags </H1>

<tsx:dbconnect  id="conn"
      userid="itso" passwd="itso"
      url="jdbc:db2:sample"
      driver="COM.ibm.db2.jdbc.app.DB2Driver">
</tsx:dbconnect>

<tsx:dbquery id="dept" connection="conn" >
    SELECT * FROM DEPARTMENT ;
</tsx:dbquery>

<tsx:repeat index="deptidx">
   <H2> <tsx:getProperty name="dept" property="DEPTNAME" /> </H2>

   <tsx:dbquery id="emp" connection="conn" >
      SELECT FIRSTNME, LASTNAME FROM EMPLOYEE
       WHERE WORKDEPT = '<tsx:getProperty name="dept" property="DEPTNO"/>';
   </tsx:dbquery>

   <tsx:repeat index="empidx">
      <LI> <tsx:getProperty name="emp" property="FIRSTNME" />
           <tsx:getProperty name="emp" property="LASTNAME" />
   </tsx:repeat>

</tsx:repeat>
</BODY>
</HTML>
```

Figure 87. Database access JSP demonstrating WebSphere tsx tags

In this example we use the *tsx:dbquery* tag to select all the departments and then all the employees within one department. For the second query we use the *deptno* property of a department in the WHERE clause. In the inner repeat loop we list the first name and last name of each employee.

Repeating over nondatabase properties

The tsx:repeat tag is not limited to iterating over the properties provided by the tsx:dbquery tag.

You can also use this tag to iterate over any indexed property within a JavaBean class or repeatedly call any method in a JavaBean until an ArrayIndexOutOfBounds exception is thrown.

Figure 88 shows a JSP that iterates over a JavaBean using a tsx:repeat tag. Figure 89 shows the source code of the JavaBean with a vector where each element is an array of two values. The bean provides two get methods to return the values from the array of a vector element.

```
<HTML>
<HEAD> <TITLE> JSP with Repeating Bean </TITLE> </HEAD>
<H1> CD Listing </H1>
<jsp:useBean id="vectorBean"
             class="itso.servjsp.jspsamples.VectorBean" scope="session" />
<tsx:repeat index="i">
   <LI> <%= vectorBean.getTitle(i) %> by
        <B> <%= vectorBean.getArtist(i) %> </B>
</tsx:repeat>
</BODY></HTML>
```

Figure 88. JSP using a bean with repeating attributes

```
package itso.servjsp.jspsamples;
public class VectorBean {
   java.util.Vector cdList = new java.util.Vector();
public VectorBean() {
   cdList.addElement( new String[] {"Woman In Me","Shania Twain"} );
   cdList.addElement( new String[] {"Come On Over","Shania Twain"} );
   cdList.addElement( new String[] {"When I Call Your Name","Vince Gill"} );
}
public String getArtist(int ix) {
   try { return ( (String[])cdList.elementAt(ix) )[1]; }
   catch (Exception e) { throw new ArrayIndexOutOfBoundsException(); }
}
public String getTitle(int ix) {
   try { return ( (String[])cdList.elementAt(ix) )[0]; }
   catch (Exception e) { throw new ArrayIndexOutOfBoundsException(); }
}}
```

Figure 89. JavaBean with repeating attributes

JSP utility example

See “Utility JSP” on page 415 in Appendix B, “Utility servlet and utility JSP” for an example of a complex utility JSP that collect useful information about the WebSphere configuration and the servlet environment.

Differences between JavaServer Page specification .91 and 1.0

The JSP 1.0 specification contains the following changes and additions over the JSP .91 specification:

- ❑Tags use XML formatting. For example, the JSP bean declaration tag *<BEAN>* is now declared using the syntax *<jsp:useBean ...>*. Similarly, WebSphere specific tags such as *<REPEAT>* are now declared using the syntax *<tsx:repeat>*.
- ❑Tags are case sensitive.
- ❑Standard tags use the mixed-case convention of Java code, for example, `jsp:useBean`.
- ❑Server-side includes (SSI) have been replaced with the *<%@ include %>* directive.
- ❑*jsp:getProperty* and *jsp:setProperty* tags have been defined.
- ❑*jsp:request* has been added, providing runtime forward and include functionality.
- ❑*jsp:include* has been added to include resources from other files.
- ❑*jsp:plugin* has been added.
- ❑Implementation of `LOOP`, `ITERATE`, `INCLUDEIF` and `EXCLUDEIF` tags have been postponed pending enhancements to the tag extension mechanism.
- ❑*<SCRIPT> </SCRIPT>* tags have been superseded with *<%! ... %>*

There have been other releases of the JSP specification such as .92 and .93. The additional functionality offered by these releases has not been discussed in this chapter.

6 WebSphere Application Server

In this chapter we describe how to use WebSphere Application Server, which throughout this chapter will be referred to as WAS.

We explain the administration environment that comes with WAS, how it is structured, and how you will use it to configure your server environment. For each of the components within the WAS topology, we show with an example how they are used in supporting the applications created in this book.

We look at where the physical files have to be placed when you are deploying applications to the WAS environment. We use our sample applications as examples for deployment.

We then discuss security in WAS and how you can use it to protect your applications and resources.

WAS overview

The WAS execution environment is shown in Figure 90.

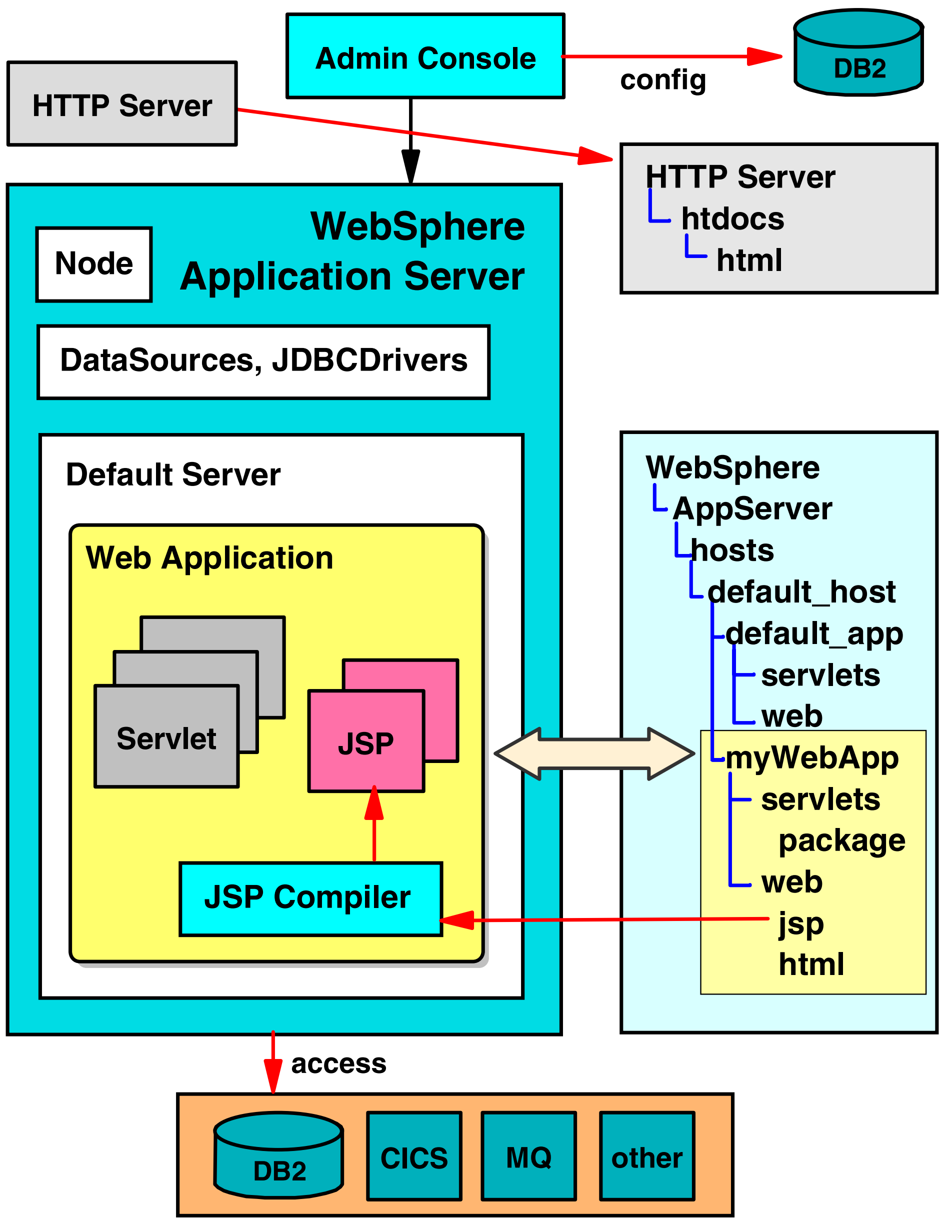


Figure 90. WebSphere Application Server execution environment

Here is a short description of the major components of WAS:

❑WAS contains a plug-in for many HTTP servers, including the IBM HTTP Server.

❑WAS runs on a node (a TCP/IP host name).

❑JDBC drivers and DataSources describe how relational databases are accessed from servlets.

❑Multiple servers (for example, the Default Server), each with a servlet engine, run on the node.

❑Servlets and JSPs are grouped into Web applications. A server contains multiple Web applications.

❑An administrative console program maintains the configuration of the Application Server in a DB2 (or Oracle) database. The configuration contains information about servers, Web applications, servlets, EJBs, the JSP compiler, DataSources, and other resources.

❑The right side of the diagram shows the directory structure for the HTTP Server and the Application Server.

The HTTP Server directory contains static HTML files.

The Application Server directory contains the directories for the Web applications, such as the default application and any user defined Web applications. For each Web application there is normally a servlets and a web subdirectory.

- The servlets subdirectory contains executable code for servlets and JavaBeans.
- The web subdirectory contains HTML and JSP files. HTML files are served to a browser by a special file handling servlet. JSP files are compiled into servlets the first time they are invoked.

❑The code in Web applications can access enterprise resources, such as relational databases (DB2 and others), CICS, MQSeries, IMS, SAP, and others. This is normally done using the Common Connector Framework and WebSphere connection pools.

WAS administration

The WebSphere administrative model allows you to:

- ❑Configure applications and their components
- ❑Control access to applications (security)
- ❑Perform daily administrative operations
- ❑Analyze usage statistics and optimize performance

WAS provides centralized administration of all your components with an administrative server tracking all of a domain's[1] contents and activities in an administrative repository. Details on how to start up a WebSphere AdminServer can be found in "Starting the WS AdminServer service" on page 30.

The administrative repository

The repository is the database of information about an administrative domain, and all its resources. Each resource in a WebSphere administrative domain corresponds to an object in the repository. So for every Web application you add to the domain, a matching Web application object, containing descriptive information about that resource, is created in the repository.

Administration servers on different machines can all access the same repository, allowing you to administer the domain from any machine.

The WebSphere Administrative Console

The WebSphere Administrative Console is a graphical administrative client that enables you to makes requests to an administrative server to access and make changes to resources in the domain. For example, you can start, stop, ping, and modify application servers in the WebSphere Administrative Console, which in turn invokes methods on the resource beans for the application servers.

For a guide on to how to start the console, see "Starting the Administrative Console" on page 31.

[1] An administrative domain can be comprised of one or more administrative servers who share an administrative repository.

Navigating the console

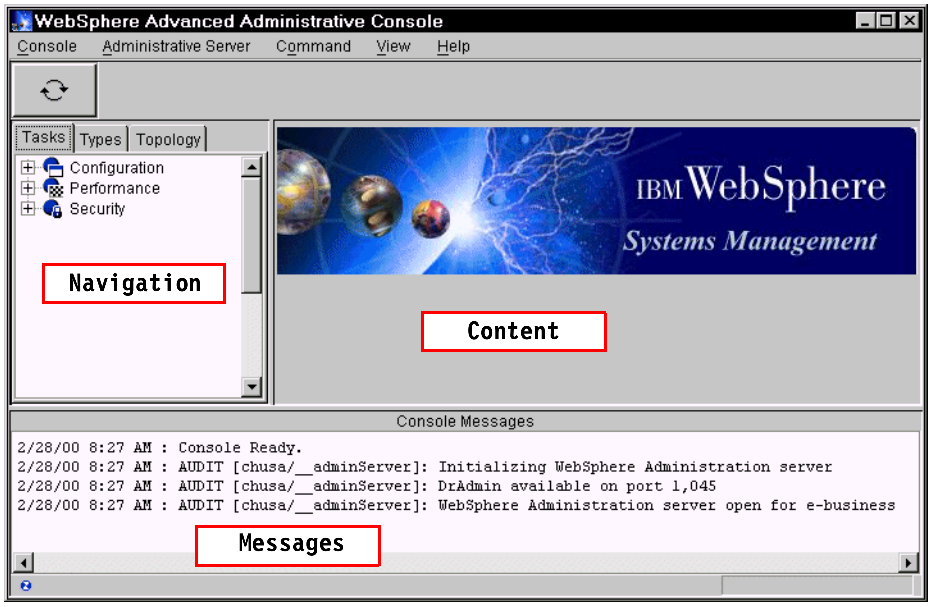
The console has three main areas (Figure 91):

❑The navigation pane on the left, with Tasks, Types, and Topology pages.

❑The content pane on the right, for displaying information based on what has been selected in the navigation pane.

❑The messages pane on the bottom, for displaying high-level messages of important events.

Each of these panes can be resized appropriately (although you may find that the console is occasionally a little stubborn when trying to do this).



```
2/28/00 8:27 AM : Console Ready.
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: Initializing WebSphere Administration server
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: DrAdmin available on port 1,045
2/28/00 8:27 AM : AUDIT [chusa/__adminServer]: WebSphere Administration server open for e-business
```

Figure 91. Navigating the WAS console

Navigation Pane

The navigation pane enables you to control what appears in the content pane. It has three different views which are accessible by selecting on the tabs at the top of the pane. Selecting any of the items within each view brings up relevant information, and sometimes editable fields, in the contents pane. The different tabs are:

❑The *Tasks* tab is for performing administrative tasks, such as creating new Web applications.

- ❑The *Types* tab is for specifying defaults and taking inventory of your components.
- ❑The *Topology* tab is primarily for surveying and managing existing components, although you can create new ones here also.

We will be using elements within each of these different views throughout this chapter.

WAS Topology

We now discuss the topology of the WAS environment, which can be accessed in the console by selecting the *Topology* tab. The topology within WebSphere is built on a containment hierarchy which can be seen in Figure 92.

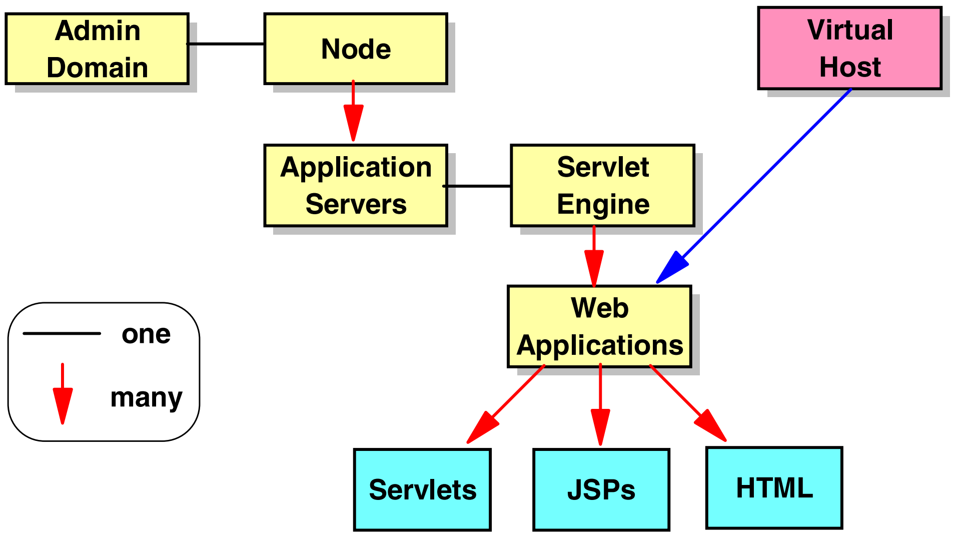
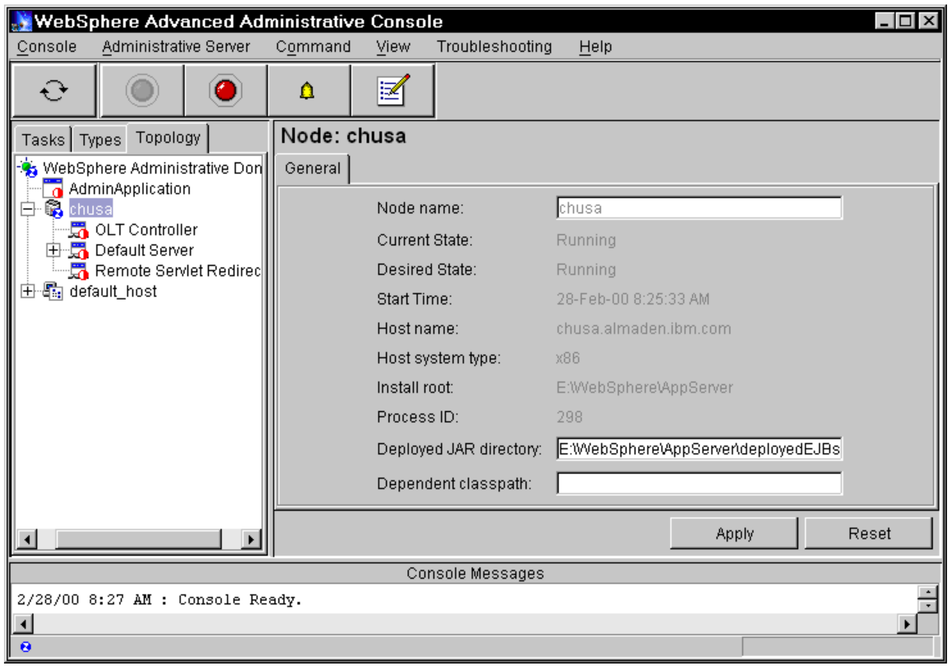


Figure 92. Topology in WAS

Node

A node represents a physical machine. After installation, WAS will have created a node representing your machine, named after the machine's host name. In Figure 93, our node is named *chusa.almaden.ibm.com* after the machine where WAS was installed.



Node: chusa

Node name:	chusa
Current State:	Running
Desired State:	Running
Start Time:	28-Feb-00 8:25:33 AM
Host name:	chusa.almaden.ibm.com
Host system type:	x86
Install root:	E:\WebSphere\AppServer
Process ID:	298
Deployed JAR directory:	E:\WebSphere\AppServer\deployedEJBs
Dependent classpath:	

```
2/28/00 8:27 AM : Console Ready.
```

Figure 93. Viewing a node in WAS

You can have more than one node, each of which would represent different machines to which you can distribute various resources. In this book, we will keep it simple and stick with the default node provided.

Application server

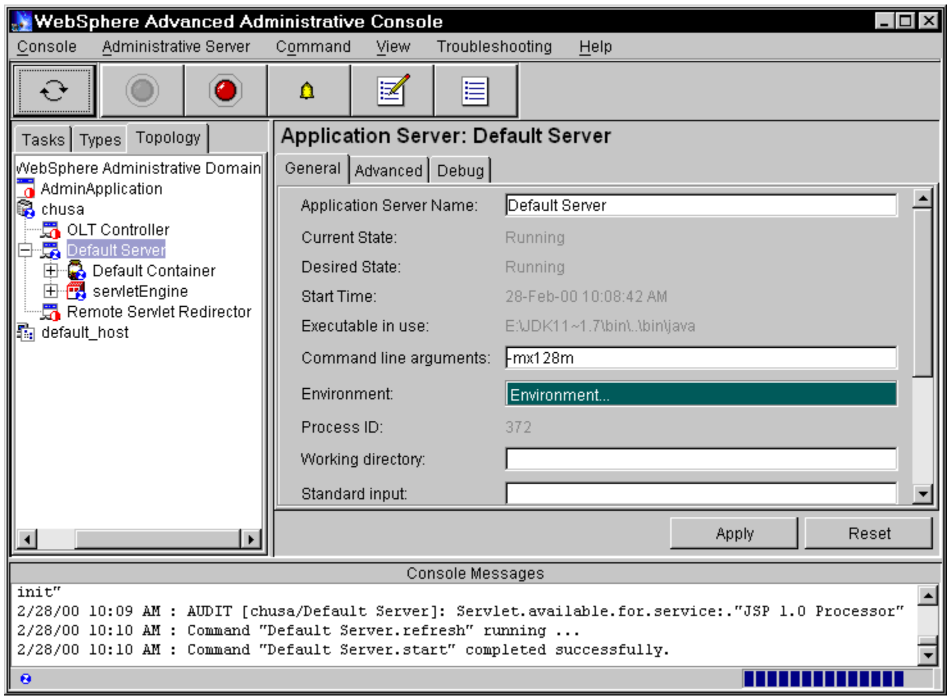
Application servers are used to extend the capabilities of a Web server to handle requests for servlets, enterprise beans and Web applications. It is important to note that the IBM WebSphere Application Server product is more than just an *application server*, and can actually be used to support multiple application server processes.

An application server in WAS has two main components:

❑A Java virtual machine configuration

❑Support for a servlet engine to handle servlet requests

After installation, WAS is configured with the default application server, appropriately named *Default Server* (Figure 94). We used this application server exclusively in our examples.



Application Server Name: Default Server
Current State: Running
Desired State: Running
Start Time: 28-Feb-00 10:08:42 AM
Executable in use: E:\JDK11~1.7\bin\..\bin\java
Command line arguments: -mx128m
Environment: Environment...
Process ID: 372
Working directory:
Standard input:

```
init"
2/28/00 10:09 AM : AUDIT [chusa/Default Server]: Servlet.available.for.service:."JSP 1.0 Processor"
2/28/00 10:10 AM : Command "Default Server.refresh" running ...
2/28/00 10:10 AM : Command "Default Server.start" completed successfully.
```

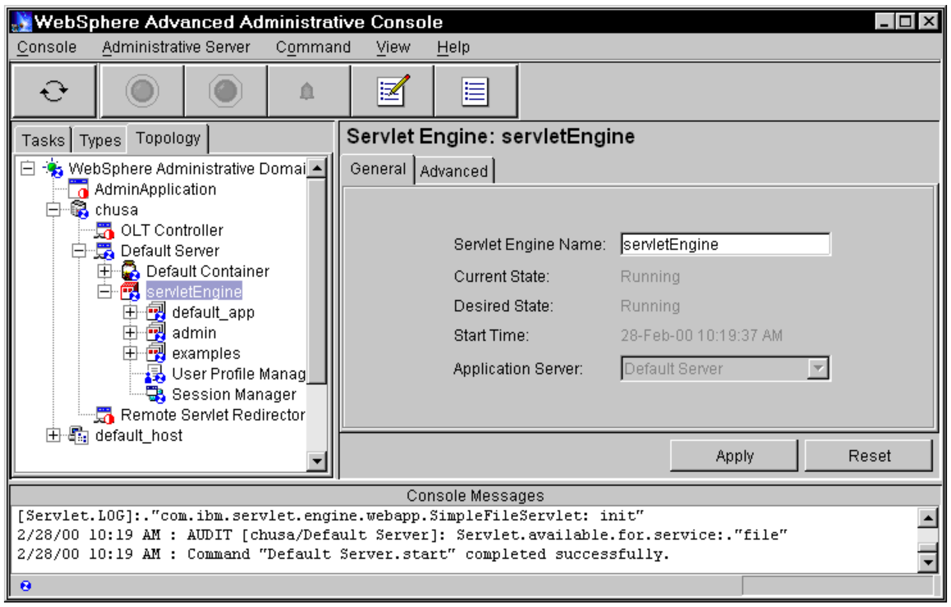
Figure 94. Viewing an application server in WAS

The application server can be started and stopped by clicking on the "green light" and "red light" buttons in the tool bar.

Servlet engine

A servlet engine is a program that runs within the application server and handles the requests for servlets, JavaServer Pages, and other types of server-side include coding. The servlet engine is responsible for creating instances of servlets, initializing them, acting as a request dispatcher, and maintaining servlet contexts for use by your Web applications.

WAS only supports one servlet engine per application server. For the purposes of the examples in this book, we will be using the one that is created by default for our application server and is named *servletEngine* (Figure 95).



Servlet Engine Name: servletEngine
Current State: Running
Desired State: Running
Start Time: 28-Feb-00 10:19:37 AM
Application Server: Default Server

```
[Servlet.LOG]:."com.ibm.servlet.engine.webapp.SimpleFileServlet: init"
2/28/00 10:19 AM : AUDIT [chusa/Default Server]: Servlet.available.for.service:."file"
2/28/00 10:19 AM : Command "Default Server.start" completed successfully.
```

Figure 95. Viewing a servlet engine in WAS

Web application

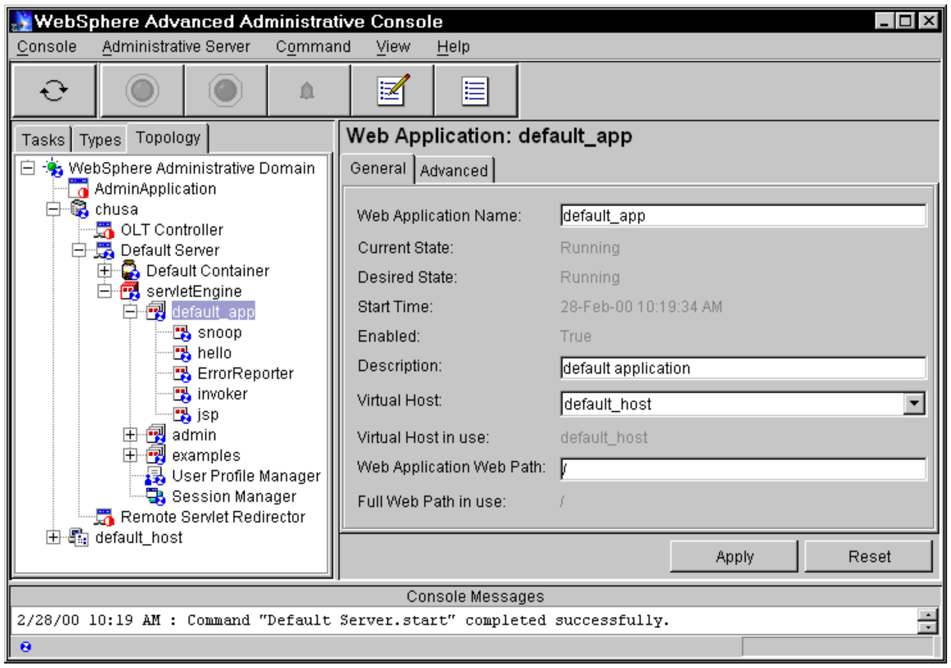
A Web application represents a grouping of servlets, JSPs, and their related resources. Managing these elements as a unit allows you to stop and start servlets in a single step. You can also define a separate document root and class path at the Web application level, thus allowing you to keep different Web applications separate in the file system.

Servlets that are running within a Web application share the same servlet context with others in the same application, allowing them to communicate with each other.

Installation of WAS creates three Web applications under the default servlet engine, and each comes with a number of default servlets. For a list of some of the common servlets that are included (and that you may also want to include in your own Web applications), see “Internal servlets” on page 134.

- ❑The *default_app* Web application (Figure 96) can be used to deploy simple servlets for testing. It has been designed to ease the migration of servlets and applications from WAS version 2. You can also use the default_app as a template for your own Web applications.
- ❑The *admin* Web application is used by WAS to install the AdminServer GUI, and you will not normally have to change it.

❑The *examples* Web application contains a few sample servlets that you can run from day-one to test your environment and give you an idea of some basic designs. You can invoke these samples using the URL `http://yourHostName/webapp/examples/`.



Web Application Name: default_app
Current State: Running
Desired State: Running
Start Time: 28-Feb-00 10:19:34 AM
Enabled: True
Description: default application
Virtual Host: default_host
Virtual Host in use: default_host
Web Application Web Path: /
Full Web Path in use: /

```
2/28/00 10:19 AM : Command "Default Server.start" completed successfully.
```

Figure 96. Default Web application in WAS

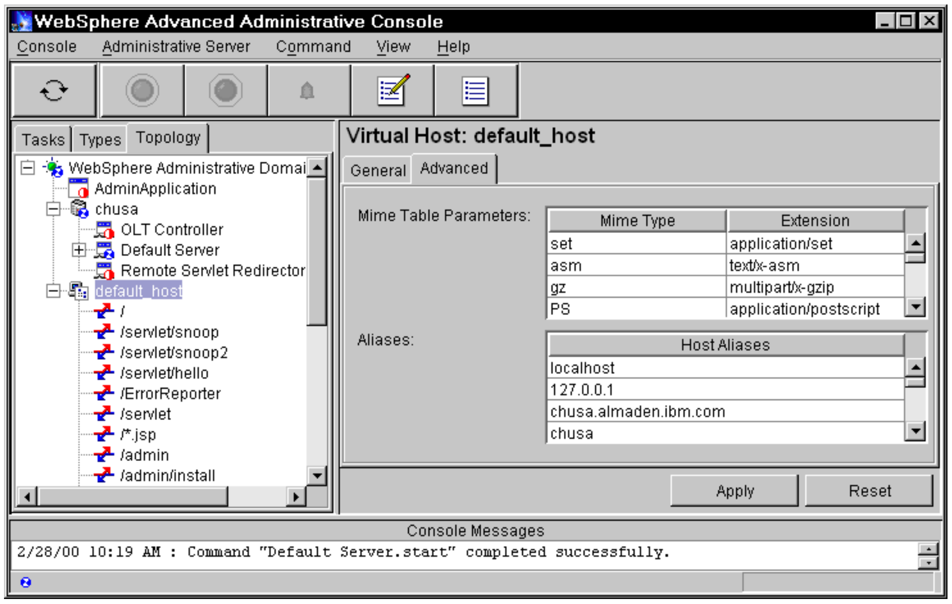
On the Advanced page of the Web application, you will find the file locations for documents (HTML, JSP) and servlets. We will show these when we define our own Web application in “Creating your own Web application” on page 135.

Virtual host

A virtual host is a mechanism allowing a single physical machine to resemble multiple host machines. Different resources, including servlets, JSPs, and Web applications, are associated with a single virtual host, and are not shared with other virtual hosts, even if they are on the same physical machine. You can also specify MIME type support at a virtual host level. This might be a common setup for an ISP who has one physical machine managing sites for a number of different customers who would not want their data visible to others.

Each virtual host has a logical name and a list of DNS aliases by which it is known, for example, *yourHostName:80*. When a servlet request is made, the server name and port number entered are compared to the list of aliases. Once a match is located with a virtual host, the servlet is then found and served up. If no match is found, an error is returned to the browser.

WAS comes configured with a default virtual host, *default_host*, with some common aliases, such as the machine's IP address, short host name, and fully qualified host name. All the default Web applications and their resources are set up to use this virtual host. This configuration is fine for the purposes of most examples in this book, and is shown in Figure 97.



Mime Table Parameters:

Mime Type	Extension
set	application/set
asm	text/x-asm
gz	multipart/x-gzip
PS	application/postscript

Aliases:

Host Aliases
localhost
127.0.0.1
chusa.almaden.ibm.com
chusa

```
2/28/00 10:19 AM : Command "Default Server.start" completed successfully.
```

Figure 97. Virtual host in WAS

For the sample application described in Part 2, we defined a separate virtual host for the Web application that uses the secure HTTPS protocol. For more information on how to set up another virtual host, refer to *Managing virtual hosts* in the Administration Console documentation that comes with the WAS product.

Internal servlets

The servlets shown in Table 12 are provided by WAS in the default_app and can also be loaded as part of your own Web application. You must add some of these servlets to your Web application for file serving from WAS directories and to compile JSPs.

Table 12. Internal Servlets for WAS

Function	Class	Additional Information
Invoke a servlet by class name	com.ibm.servlet.engine. webapp. Invoker	In addition to invoking the servlet by the servlet Web paths configured via the Administrative Console, the Invoker servlet enables you to invoke servlets by their class names. Using the Invoker servlet is considered a security exposure.
Serve HTML files in the application's document root using the Web application prefix	com.ibm.servlet.engine. webapp. SimpleFileServlet	This servlet handles files in the application document root whose URLs are not covered by the HTTP server configuration through pass rules.
Enable the JSP 0.91 page compiler	com.ibm.servlet.jsp.http. pagecompile. PageCompileServlet	This servlet is in ibmwebas.jar.
Enable the JSP 1.0 page compiler	com.sun.jsp.runtime. JspServlet	This servlet is in jsp10.jar. See the Sun JSP 1.0 specification for more information.
Use the extended error reporting function	com.ibm.servlet.engine. webapp. DefaultErrorReporter	Use this servlet if you want error reporting through an error page, but you do not want to write your own error page.
Enable a servlet chain	com.ibm.websphere. servlet.filter. ChainerServlet	Use this servlet for chaining multiple servlets together.

See “Adding JSP support to a Web application” on page 147 for more information on how to set up the JSP compiler at 0.91 or 1.0 specification.

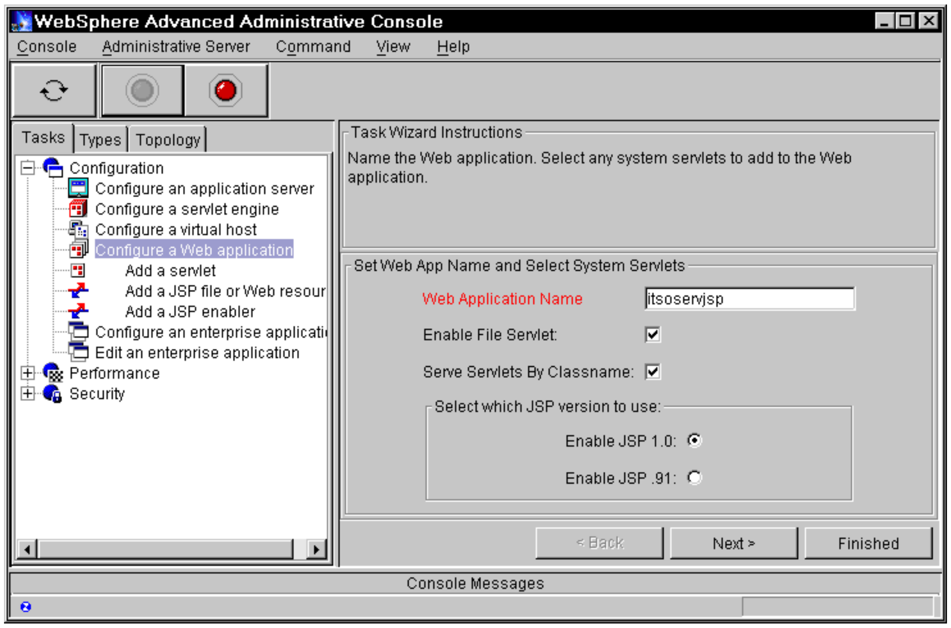
Creating your own Web application

We suggest that you create a Web application for all the servlets and JSPs that have to work together. For this redbook we created our own Web application.

Using the Task Wizard

When creating and adding new resources to your WAS setup, you can generally use one of the task wizards to help you through the process. These can be found by selecting the *Task* tab in the Administrative Console.

Select the wizard you want to use—in our case this is *Configure a Web application*—and click on the green *Start Task* button (Figure 98). The wizard appears in the right-hand pane and prompts you for the name of the new Web application, for example, *itsoservjsp*. You then select which internal servlets you would like pre-loaded (for more information see “Internal servlets” on page 134) and the level of JSP support. For more information on setting up JSP support, see “Adding JSP support to a Web application” on page 147.


WebSphere Advanced Administrative Console
Console
Administrative Server
Command
View
Help
Tasks
Types
Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resour
Add a JSP enabler
Configure an enterprise applicati
Edit an enterprise application
Performance
Security
Task Wizard Instructions
Set Web App Name and Select System Servlets
< Back
Next >
Finished
Console Messages

Name the Web application. Select any system servlets to add to the Web application.

Web Application Name: itsoservjsp

Enable File Servlet: ✓

Serve Servlets By Classname: ✓

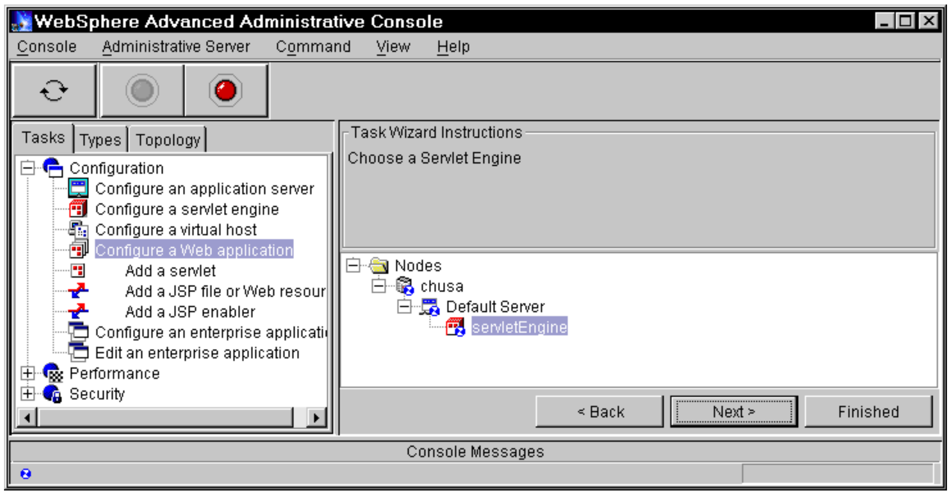
Select which JSP version to use:

Enable JSP 1.0: ●

Enable JSP .91: ○

Figure 98. Configuring a Web application: name, servlets, JSP support

Click on *Next* and you are prompted to select a servlet engine. Our configuration has only one node with one application server (*Default Server*), therefore we can only select the *servletEngine* belonging to the default server. Expand the node until you find the servlet engine, select it, and click on *Next* (Figure 99).



Choose a Servlet Engine

Figure 99. Configuring a Web application: servlet engine

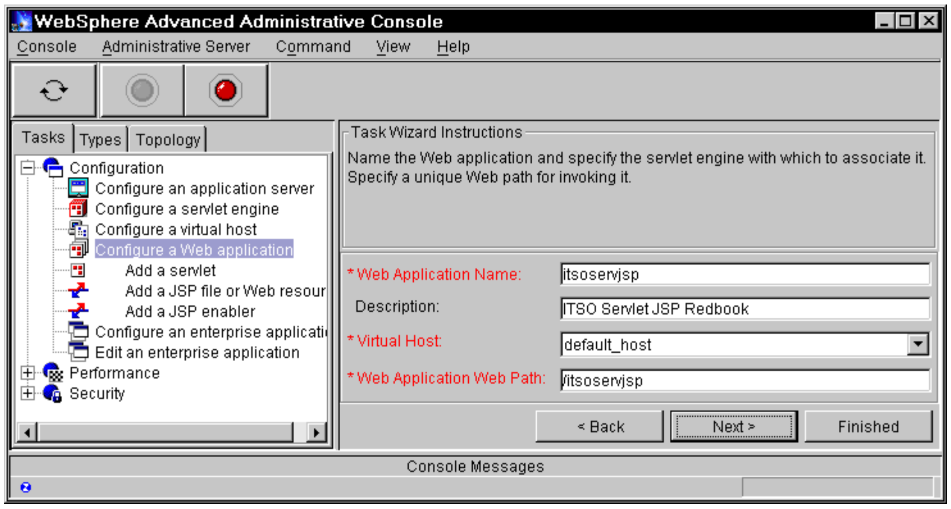
The next step prompts you for the virtual host and the Web application Web path. The Web path, when combined with the virtual host, is the base URL that is used in Web browsers to locate a resource within the Web application (Figure 100).

For instance, if you wanted to access the *snoop* servlet that was part of a Web application with a Web path of `/webapp/itsoservjsp` and that Web application was on a virtual host with an alias of *chusa.almaden.ibm.com*, then the correct URL would be:

```
http://chusa.almaden.ibm.com/webapp/itsoservjsp/snoop
```

For our Web application we select the default virtual host (*default_host*), however, we set up the Web path as */itsoservjsp* (the default would be /webapp/itsoservjsp). Therefore, our URL for servlets and JSP will be:

```
http://chusa.almaden.ibm.com/itsoservjsp/..servletname...
```


WebSphere Advanced Administrative Console
Console Administrative Server Command View Help
Tasks Types Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resour
Add a JSP enabler
Configure an enterprise applicati
Edit an enterprise application
Performance
Security
Task Wizard Instructions

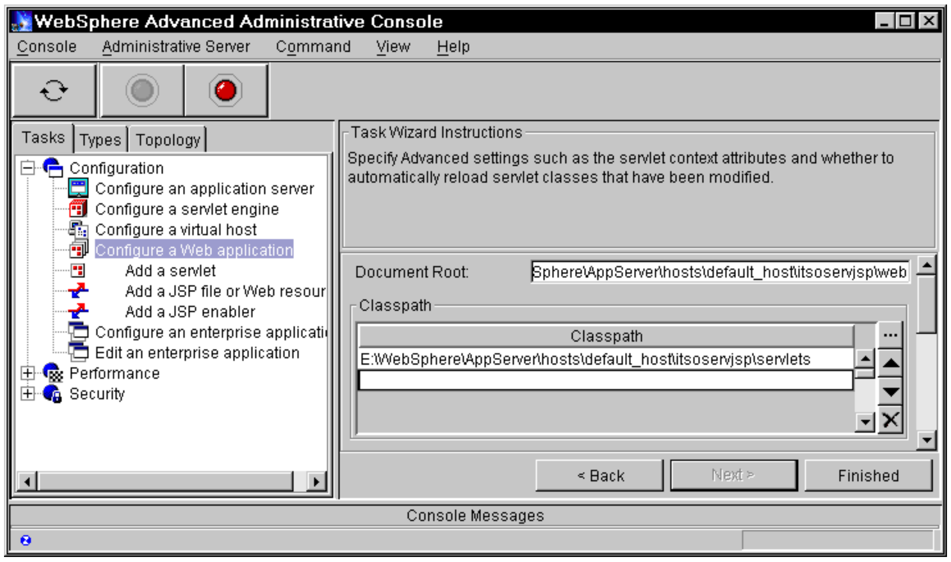
Name the Web application and specify the servlet engine with which to associate it. Specify a unique Web path for invoking it.

* Web Application Name: itsoservjsp
Description: ITSO Servlet JSP Redbook
* Virtual Host: default_host
* Web Application Web Path: /itsoservjsp

< Back
Next >
Finished
Console Messages

Figure 100. Configuring a Web application: virtual host and Web path

The next step prompts you for the document root and the class path (for servlets) of the Web application (Figure 101). The document root is where all your document files used in this Web application reside; this includes HTML files and JSPs. For more information on the class path and class loading in general, see “Class loading and reloading” on page 142.


WebSphere Advanced Administrative Console
Console Administrative Server Command View Help
Tasks Types Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resour
Add a JSP enabler
Configure an enterprise applicati
Edit an enterprise application
Performance
Security
Task Wizard Instructions

Specify Advanced settings such as the servlet context attributes and whether to automatically reload servlet classes that have been modified.

Document Root: Sphere\AppServer\hosts\default_host\itsoservjsp\web

Classpath

Classpath
E:\WebSphere\AppServer\hosts\default_host\itsoservjsp\servlets

< Back
Next >
Finished
Console Messages

Figure 101. Configuring a Web application: document root and class path

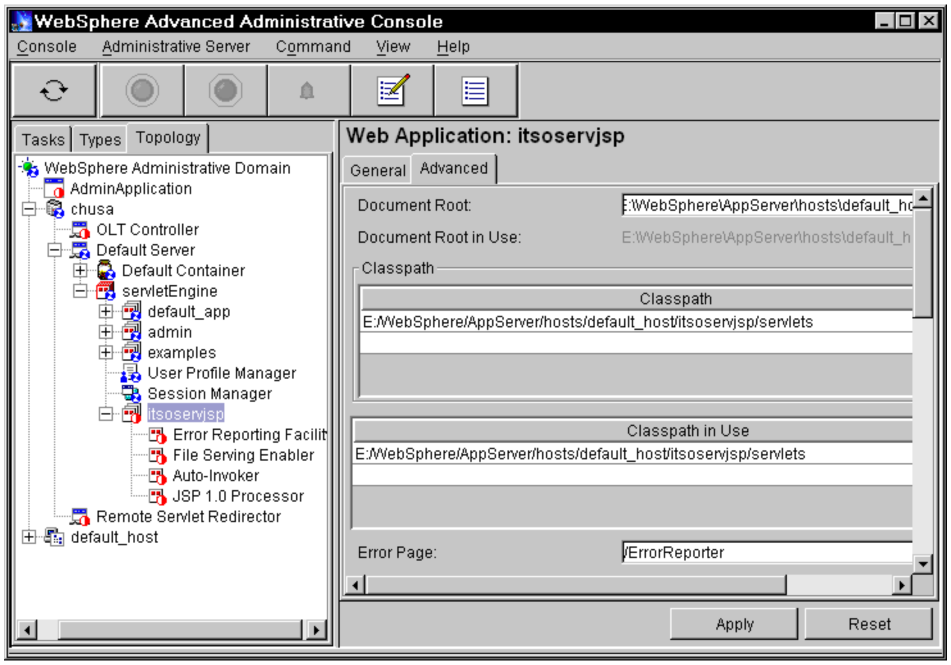
The directories that are proposed are based on the Web path specified in the previous step, in our case:

```
d:\WebSphere\AppServer\hosts\default_host\itsoservjsp\web
d:\WebSphere\AppServer\hosts\default_host\itsoservjsp\servlets
```

These directories are not automatically created for you, so you will have to do this as a manual step afterwards (see “Creating the required Web application directories” on page 139). For now, leave the proposed values and click on *Finished*. Check the messages pane for a confirmation: *Command "WebApplication.create" completed successfully.*

Setting up your default error page

In the Administrative Console, click on the *Topology* tab and navigate down the tree until you see your newly created Web application (Figure 102).



Document Root: E:\WebSphere\AppServer\hosts\default_hc
Document Root in Use: E:\WebSphere\AppServer\hosts\default_h
Classpath: E:/WebSphere/AppServer/hosts/default_host/itsoservjsp/servlets
Classpath in Use: E:/WebSphere/AppServer/hosts/default_host/itsoservjsp/servlets
Error Page: /ErrorReporter

Figure 102. Viewing a newly created Web application

Notice that in addition to the three servlets we set up with the wizard, another servlet called *ErrorReporter* has appeared. This is what the Web application uses for handling errors in JSPs and servlets.

If you want to use your own error page instead, change the value on the *Advanced* page, for example, enter *error.jsp*, which can be found in the document root of *default_app*. However, for the purposes of our application, we leave it set to the *ErrorReporter*, as it does all we need.

Creating the required Web application directories

For every new Web application that you create, you probably have to create the physical directories for the documents and Java classes (servlets).

Underneath the `Websphere/AppServer/hosts/default_host` directory is where each of the Web applications that are part of the *default_host* has a directory. For each Web application directory, you normally have a *servlets* directory for the class path and a *web* directory for the document root. These three directories have to be created for our new Web application *itsoservjsp* (Figure 103).

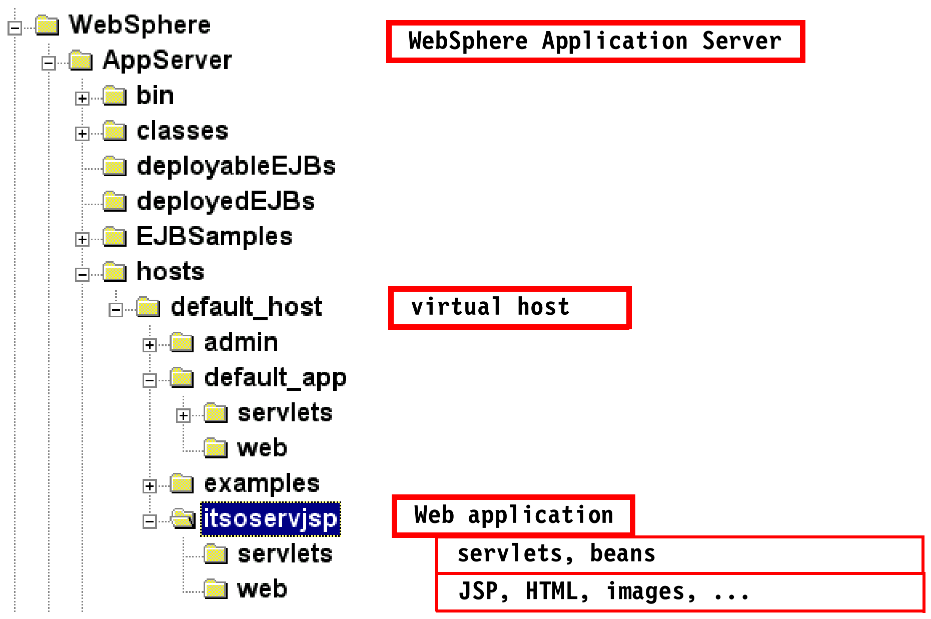


Figure 103. Web application directory structure

For simple tests you can copy the *SnoopServlet.class* and the *very_simple.jsp* files from the default_app directories to the itsoservjsp directories. Another nice test servlet is the *ServletEngineConfigDumper.class* from the *examples\servlet* directory.

Deploying files to WAS

When you have developed an application in your test environment, you will at some stage want to deploy that application to the WAS environment.

After creating the necessary components—such as a Web application—in the WS Admin Console interface, you have to physically put the resource files into the correct directories. Table 13 displays where particular files must reside.

Table 13. WebSphere application directories

Description	File extension	Directory path
HTML documents and related files	html, .shtml, .jhtml, .gif, .jpg, .au, and so forth	The Web application document root (or under the HTTP server root)
JavaServer Pages	.jsp	The Web application document root
Servlets, JavaBeans, and other Java classes	.class, .jar, .ser	The application class path or the Application Server class path (for classes that are not to be reloaded, such as serialized objects and servlets that use Java Native Interface methods)[a]. If the servlets are in a package, mirror the package structure as subdirectories under the application class path.
Servlet configuration file	.servlet	The directory that contains the servlet

a. See “Class loading and reloading” on page 142

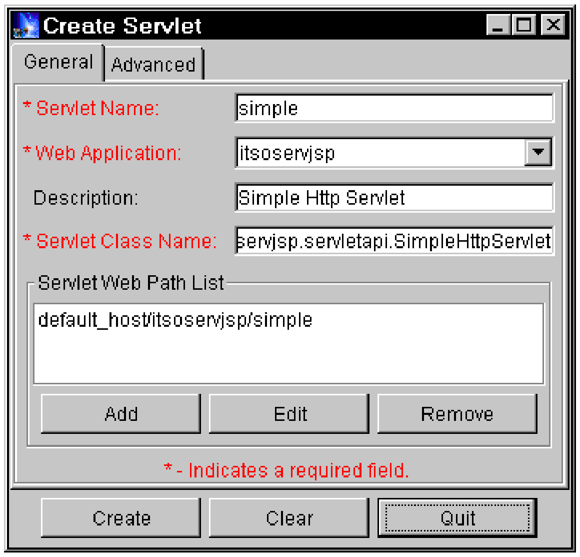
For more information on how to deploy Java classes from VisualAge for Java into specific directories, see “Importing and exporting code” on page 172. For information on how to deploy files of all types from WebSphere Studio into specific directories, see “Project relationships and integrity” on page 253.

Defining servlets

We set up the Web application in a way that servlets can be invoked by class name. WAS also enables us to invoke servlets by an alias name, and this is the preferred technique.

Let us take the simple HTTP servlet of Figure 34 on page 48. We deploy the servlet to `WebSphere\AppServer\hosts\default_host\itsoservjsp\servlet` into the subdirectory `itso\servjsp\servletapi`.

We define the servlet in WAS by selecting the itsoservjsp Web application in the Topologies pane and selecting *Create -> Servlet* from the context menu. We enter *simple* as the servlet name, itsoservjsp as the Web application (prefilled), a short description, and *itso.servjsp.servletapi.SimpleHttpServlet* as the class name. Click on *Add* and enter */itsoservjsp/simple* as the servlet Web path (this is the alias to be used in the browser). Click on *Create* to define the servlet (Figure 104). This action adds the servlet to the list of servlets under the itsoservjsp Web application.



* Servlet Name: simple
* Web Application: itsoservjsp
Description: Simple Http Servlet
* Servlet Class Name: servjsp.servletapi.SimpleHttpServlet
Servlet Web Path List
default_host/itsoservjsp/simple
* - Indicates a required field.

Figure 104. Creating a servlet for the Web application

Start the Web application

If the application server is already running, you can start the new Web application from the console. Right-click on the Web application and select *Restart Web App*. The Web application is also started when the application server is started or restarted.

Test the sample servlets

Open a browser and enter the following URLs:

```
http://localhost/itsoservjsp/simple
http://localhost/itsoservjsp/very_simple.jsp
http://localhost/itsoservjsp/servlet/SnoopServlet
http://localhost/itsoservjsp/servlet/ServletEngineConfigDumper
```

Class loading and reloading

Class loading and automatic reloading in the WAS environment has been written to help keep all the Web application components synchronized when there are any code changes.

A Web application's scope is its application class path plus the system class loader class path. When you configure a Web application, you specify its class path, which contains the servlets and the non-servlet Java components, such as JavaBeans that are used in the servlets. Whenever one of the loaded classes in that class path has been changed, *all* of the classes in that class path are reloaded. This helps to keep the Java components synchronized.

There will be occasions where you do not want certain classes to be reloaded, and you can prevent this from happening by adding those classes to the application server class path instead of the application class path. The classes are then not reloaded, although the objects will be.

Java components that should not have their classes reloaded are:

❑Java objects that are added to sessions because they are serialized.

❑Java classes that call Java Native Interface (JNI) methods.

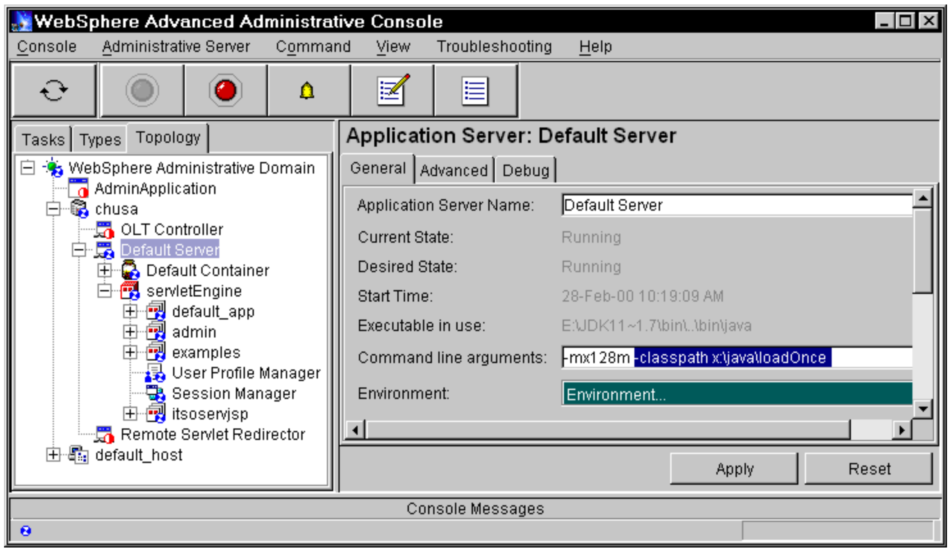
❑Objects passed as arguments for remote calls.

Changing the application server class path

The application server class path is automatically set when you install WAS. The default setting for the class path contains all of the Application Server APIs (the JAR files in the `d:\WebSphere\AppServer\lib` directory). When the Application Server starts, the system class loader automatically loads the classes in the application server class path. Classes in this class path are not reloaded.

If you want to add individual classes or jar files to the application server class path, you set up a command-line argument (see Figure 105).

You then have to restart the application server, which adds the additional directories (in this case `d:\java\loadOnce`) to the system class loader class path.



Application Server: Default Server

Application Server Name: Default Server
Current State: Running
Desired State: Running
Start Time: 28-Feb-00 10:19:09 AM
Executable in use: E:\JDK11~1.7\bin\..\bin\java
Command line arguments: -mx128m -classpath x:\java\loadOnce
Environment: Environment...

Figure 105. Updating the Application Server class path

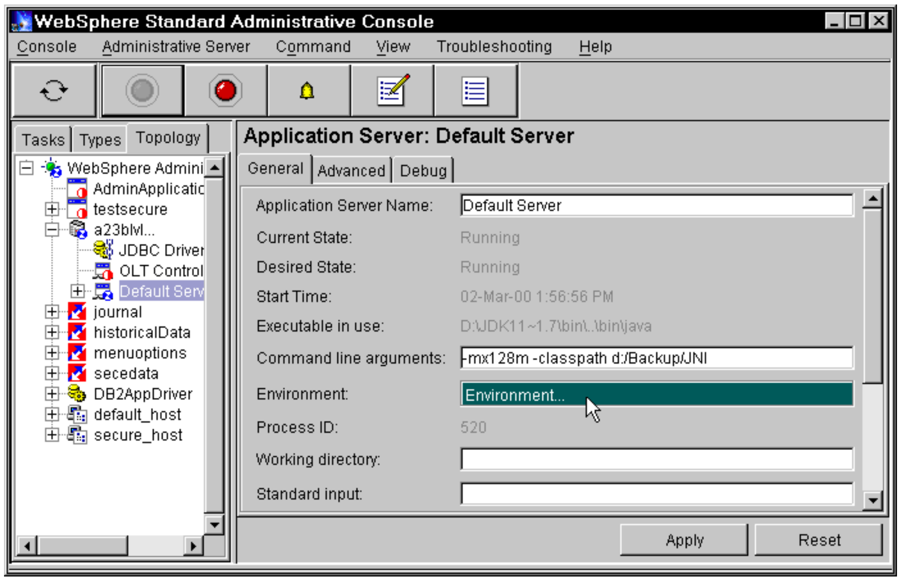
Using JNI in WAS

If your application needs to use JNI, then there are two configuration steps that need to be completed:

1. Add the jar file containing your JNI classes to the application server class path (see "Changing the application server class path" on page 142 for instructions on how to do this).
2. Create a *path* environment variable for the application server that points at the location of the relevant DLL files.

Creating an application server environment variable

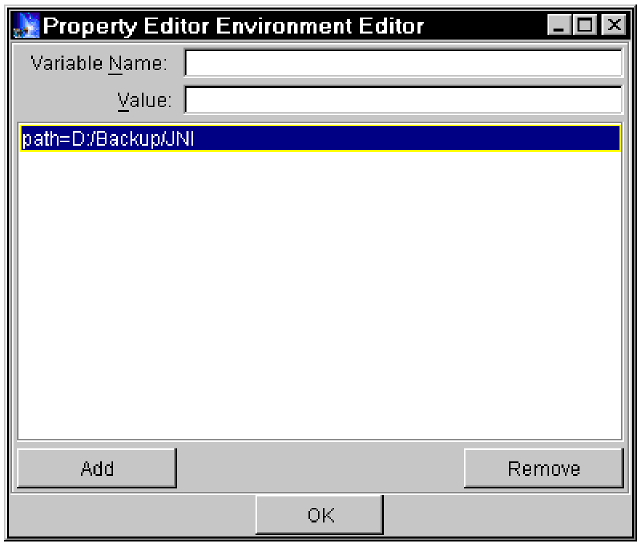
To create an environment variable for an application server, select the application server within the Topology view and make sure the *General* tab is selected in the right-hand pane (Figure 106).



Application Server Name: Default Server
Current State: Running
Desired State: Running
Start Time: 02-Mar-00 1:56:56 PM
Executable in use: D:\JDK11~1.7\bin\..\bin\java
Command line arguments: -mx128m -classpath d:/Backup/JNI
Environment: Environment...
Process ID: 520
Working directory:
Standard input:

Figure 106. Locating the environment variables for an application server

Click on the *Environment* field, and the *Property Editor Environment Editor* window will appear. Enter *path* for the Variable Name and the location of the DLL files as the Value (separating different locations with a semi-colon). Then click *Add* and the window should look similar to Figure 107.



Variable Name:
Value:
path=D:/Backup/JNI

Figure 107. Property Editor Environment Editor

Select *OK* to close this window. You **must** then click *Apply* to complete this change.

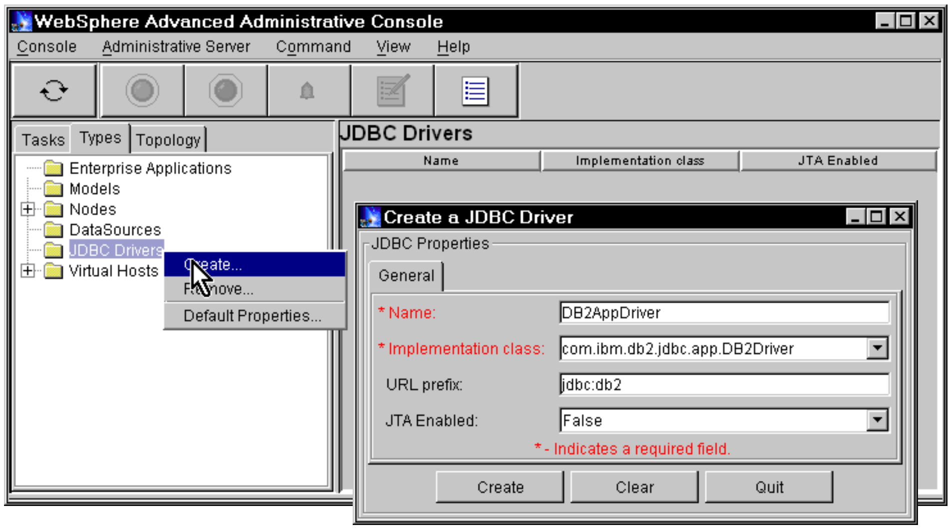
Restarting the application server should enable you to use the JNI functionality.

Setting up connection pools

WAS provides you with the ability to access databases through connection pools. In particular, you can use the DataSource mechanism to set up connection pools to particular databases, and then within your code simply ask the DataSource to pass you a connection to the database. This architecture helps to provide robustness and efficiency when dealing with database connections.

Creating a JDBC driver

To set up a DataSource, you first need to create a JDBC driver. Select the *Types* view in the left-hand pane, and right-click on *JDBC Drivers*. Select *Create* and you are prompted for the specification of the driver (Figure 108).



* Name:	DB2AppDriver
* Implementation class:	com.ibm.db2.jdbc.app.DB2Driver
URL prefix:	jdbc:db2
JTA Enabled:	False

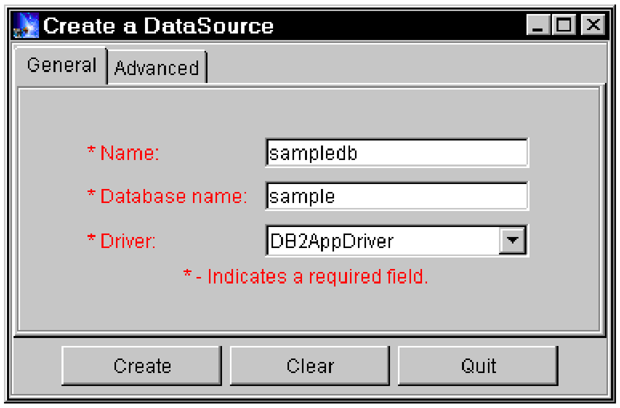
* - Indicates a required field.

Figure 108. Creating a JDBC driver

Enter a name for the driver and select the *Implementation class* from those available. Click on *Create* and the driver is created.

Creating a DataSource

Once you have created a JDBC driver, you can create a DataSource. Right-click on *DataSources* in the *Types* view, select *Create* and complete the fields in the dialog (Figure 109).



* Name: sampledb

* Database name: sample

* Driver: DB2AppDriver

* - Indicates a required field.

Figure 109. Creating a DataSource

Enter the name of the DataSource and the database name that you want to access, and then click on *Create*. The advanced page of the dialog contains parameters that set the size of the connection pool and time-out values.

You now have created a DataSource with an underlying connection pool through which you can access the SAMPLE database. For more information on how to use this, and on DataSources and connection pooling in general, you can refer to the user documentation that comes with WAS.

The JDBC driver and the DataSource are now visible at the bottom on the *Topology* page.

Migrating from the connection manager

WAS also provides support for handling connection pools through the connection manager, a facility that was available in older versions of WAS. However, for any new code that you write, we recommended that you use the new connection pool implementation, as it conforms closer to the JDBC 2.0 API standard, and the connection manager classes are deprecated in WAS Version 3.0. You should also consider migrating old database access code to use the new DataSource capabilities, because the code changes are fairly simple (refer to *Connection pooling implementation* in the user documentation).

Using JavaServer Pages in WAS

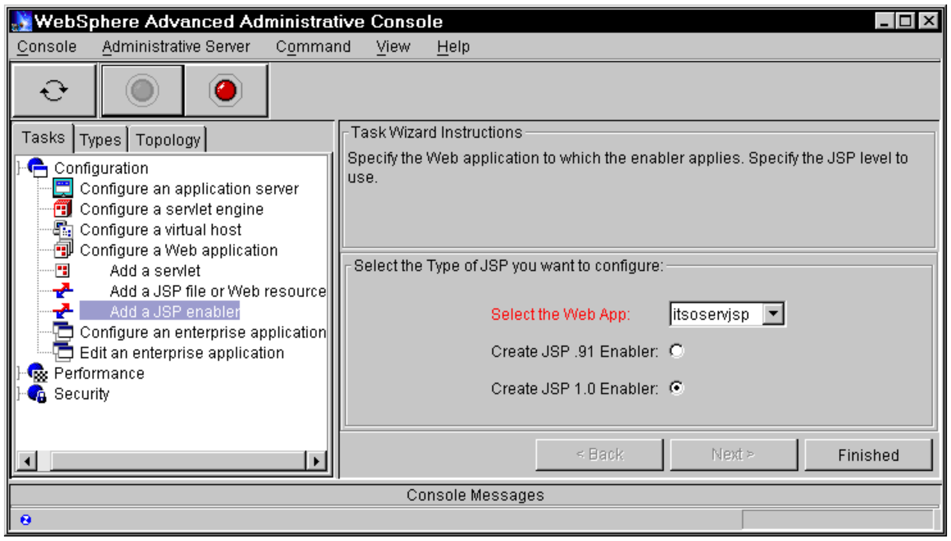
WAS supports the use of JavaServer Pages at both the 0.91 and 1.0 API levels, as well as extending the base JSP 1.0 specification (see Chapter 5, “JavaServer Pages” on page 95).

Adding JSP support to a Web application

You can configure individual Web applications to support a specific level of the JSP API by adding the appropriate JSP enabler to a Web application. If you used the Task Wizard to create the Web application, this may already have been done.

Click on the *Tasks* tab of the Administrative Console and select *Add a JSP Enabler* and click on the green *Start Task* button.

Select the Web application, the desired level of JSP support, and click on *Finished* (Figure 110). You can check that the servlet has been successfully added by locating it in the Web application under the *Topology* tab.


WebSphere Advanced Administrative Console
Console
Administrative Server
Command
View
Help
Tasks
Types
Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resource
Add a JSP enabler
Configure an enterprise application
Edit an enterprise application
Performance
Security
Task Wizard Instructions
Select the Type of JSP you want to configure:
< Back
Next >
Finished
Console Messages

Specify the Web application to which the enabler applies. Specify the JSP level to use.

Select the Web App: itsoservjsp

Create JSP .91 Enabler: ○

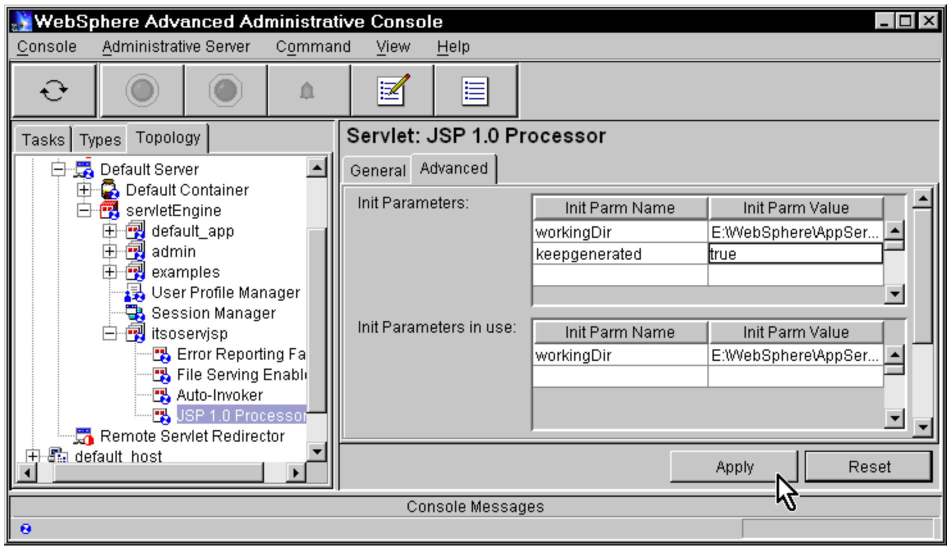
Create JSP 1.0 Enabler: ●

Figure 110. Configuring a JSP Enabler

Keeping Java source files from JSP 1.0 compilation

The JSP 1.0 enabler generates a Java source file for each JSP 1.0 file. If you want to keep the generated .java files for a JSP 1.0 page, then you need to add an extra initialization parameter to the JSP servlet.

Select the JSP 1.0 servlet in your the application, and click on the *Advanced* tab in the right-hand pane. Here you can add the initialization parameter *keepgenerated* with a value of *true* to the JSP servlet (Figure 111).



Default Server
Default Container
servletEngine
default_app
admin
examples
User Profile Manager
Session Manager
itsoservjsp
Error Reporting Fa
File Serving Enabl
Auto-Invoker
JSP 1.0 Processor
Remote Servlet Redirector
default_host

Servlet: JSP 1.0 Processor

Init Parameters:

Init Parm Name	Init Parm Value
workingDir	E:\WebSphere\AppSer...
keepgenerated	true

Init Parameters in use:

Init Parm Name	Init Parm Value
workingDir	E:\WebSphere\AppSer...

Figure 111. Adding an initialization parameter to a servlet

You have to restart the Web application to activate the change. The Java source files will be stored under the *Temp* subdirectory of the WAS installation directory:

```
d:\WebSphere\AppServer\temp\default_host\...yourwebapplication...
```

Use this option to keep the generated .java file for debugging purposes only, and empty the directory from all the Java files now and then. It is safer and more efficient not to use this option in a production environment.

Note: The temp subdirectory will contain many files. Each compilation of a JSP adds a new file with a suffix (it does not overlay the previous compilation). You should periodically remove old files from this directory.

Security

WAS provides a unified security model for both Web resources (such as servlets and JSPs), and enterprise beans. Because we are not using enterprise beans in this book, we will not be concerned with how security works for these in particular, although most of what we cover applies to enterprise beans also.

In this section, we discuss the basics of how security works within a WAS environment, and then we explain how you can set up security through the administrative console on your Web resources.

How security works in WAS

Figure 112 describes basic steps that are carried out when a user requests a Web resource that has been made secure with basic authentication.[2]

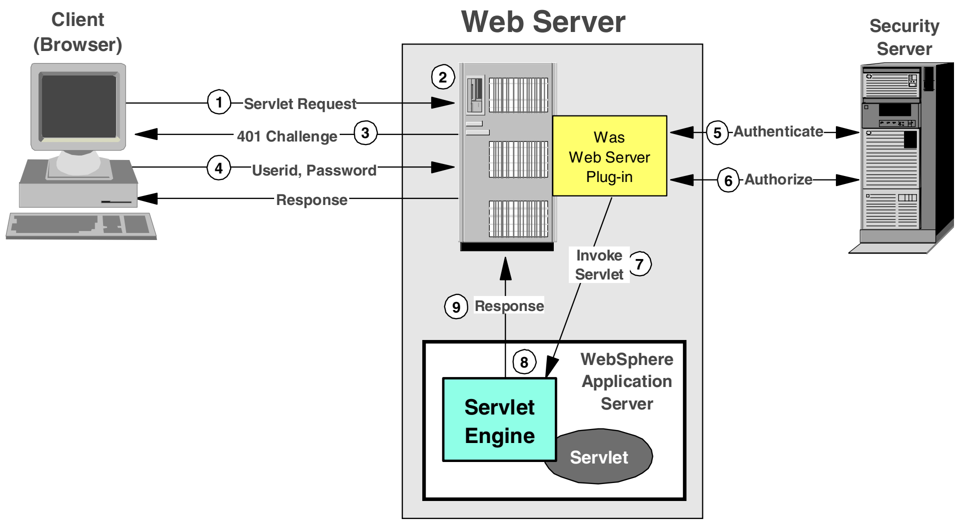


Figure 112. Basic security in WAS

1. The user requests a Web resource.
2. The Web server determines that the resource is a protected URI serviced by WebSphere.
3. The Web server issues a challenge back to the user asking them to prove who they are.

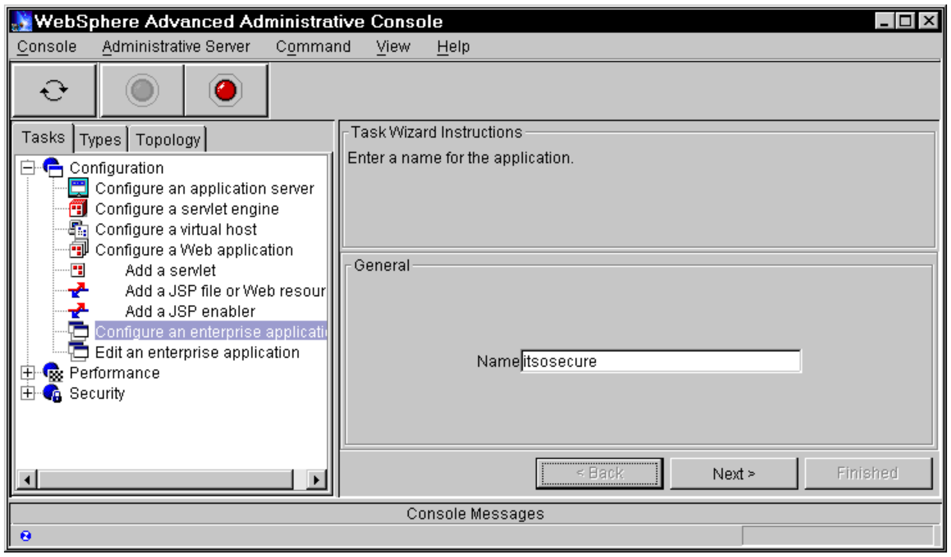
[2] There are other forms of authentication, but for now we will focus on the basic method.

4. The user responds with their user ID and password.
5. The Web server delegates this information to WebSphere's security server that authenticates the user ID and password.
6. Once the user has been authenticated, the user's permissions are consulted to see if the user is authorized to access the requested resource.
7. Upon successful authorization, a security context is set up for the request, and this is passed on to the servlet engine.
8. Upon invocation of the method (for example, *doGet*) on the resource, the user information is extracted from the security context, and the user's authorization to access that method is verified.
9. The results of the method are sent back to the user's browser.

Configuring an enterprise application

Before we can configure security in WAS, we have to define an enterprise application containing resources such as Web applications. Enterprise applications are a grouping that WAS uses for security configurations.

Select the *Configure an enterprise application* task from the Task pane, and click on the green *Start* button. You are prompted to give enter an enterprise application name, for example, *itsosecure* (Figure 113).


WebSphere Advanced Administrative Console
Console
Administrative Server
Command
View
Help
Tasks
Types
Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resour
Add a JSP enabler
Configure an enterprise applicati
Edit an enterprise application
Performance
Security
Task Wizard Instructions
General
< Back
Next >
Finished
Console Messages

Enter a name for the application.

Name itsosecure

Figure 113. Creating an enterprise application

Click on *Next*, and you are prompted to add resources to the enterprise application. Expand Web applications, select *itsoservjsp*, and click on *Add* (Figure 114).

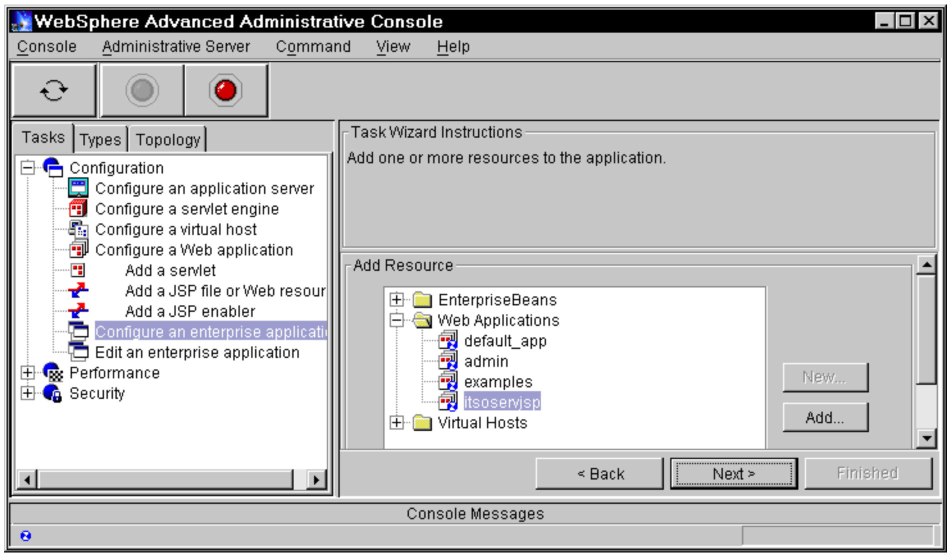
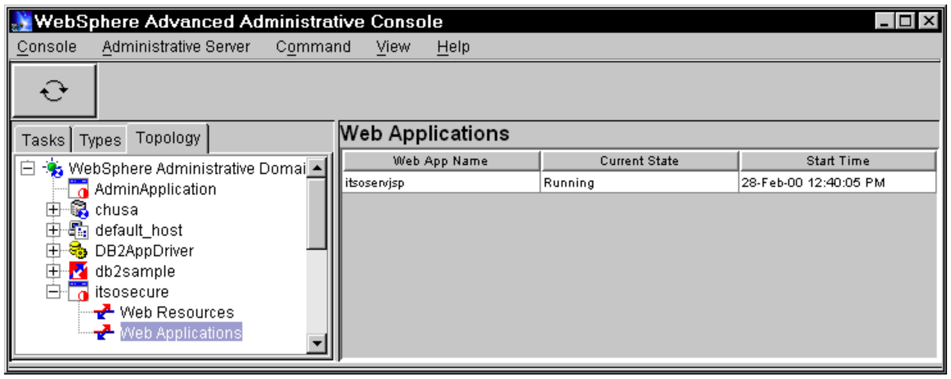

WebSphere Advanced Administrative Console
Console Administrative Server Command View Help
Tasks Types Topology
Configuration
Configure an application server
Configure a servlet engine
Configure a virtual host
Configure a Web application
Add a servlet
Add a JSP file or Web resour
Add a JSP enabler
Configure an enterprise applicati
Edit an enterprise application
Performance
Security
Task Wizard Instructions
Add one or more resources to the application.
Add Resource
EnterpriseBeans
Web Applications
default_app
admin
examples
itsoservjsp
Virtual Hosts
New...
Add...
< Back
Next >
Finished
Console Messages

Figure 114. Adding resources to an enterprise application

When you are finished adding the Web resources, click on *Next*. All enterprise application resources are listed for confirmation. Click on *Finished* and the enterprise application is built.

If you want to see your enterprise application, select the *Topology* view (Figure 115).


WebSphere Advanced Administrative Console
Console Administrative Server Command View Help
Tasks Types Topology
WebSphere Administrative Domai
AdminApplication
chusa
default_host
DB2AppDriver
db2sample
itsosecure
Web Resources
Web Applications

Web Applications

Web App Name	Current State	Start Time
itsoservjsp	Running	28-Feb-00 12:40:05 PM

Figure 115. Enterprise application topology

You can add or remove resources associated with an enterprise application with the *Edit an enterprise application* task.

Setting up security in WAS

We now discuss the steps required in setting up security in a WAS environment. Now that we have an enterprise application created, we are ready to begin setting up security.

The tasks for setting up security are listed in the Tasks pane under security (Figure 116). You have to go through all of the tasks to have basic security configured and enabled.

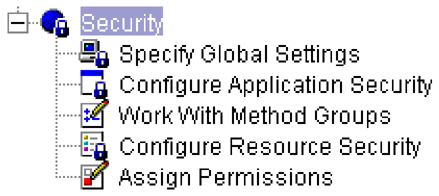


Figure 116. WAS security tasks

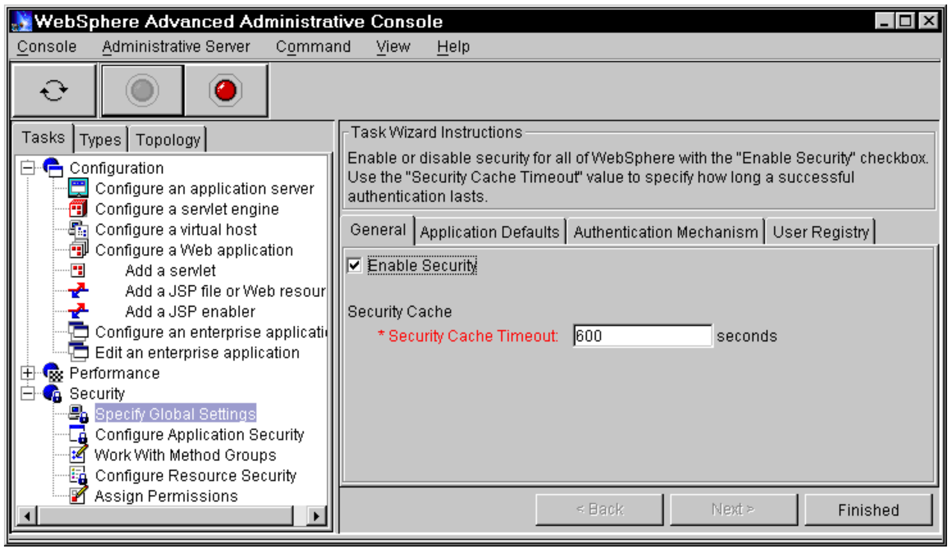
Specify global settings

You must first enable security globally for the WAS environment, and set up the default settings that all your enterprise applications will inherit.

Start the *Specify Global Settings* task under *Security* in the Tasks view. This displays a multi-tabbed view, and you will see the dialog shown in Figure 117. We will go through each tab one at a time.

General

Make sure the check box marked *Enable Security* is selected; this will turn on security for your WAS environment, although you will not yet have protected any individual resources.



Task Wizard Instructions

Enable or disable security for all of WebSphere with the "Enable Security" checkbox. Use the "Security Cache Timeout" value to specify how long a successful authentication lasts.

[✓] Enable Security

Security Cache

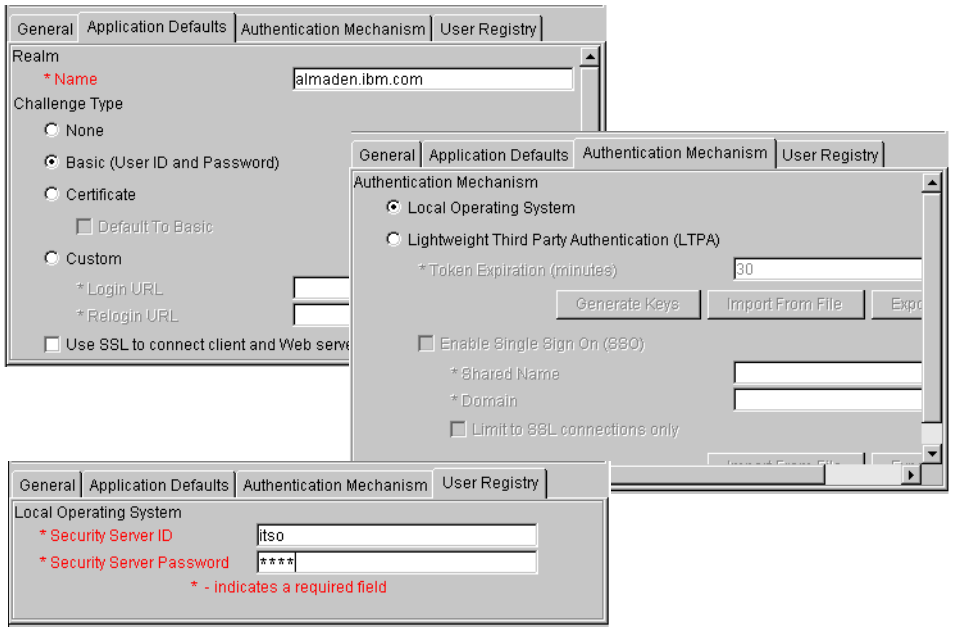
* Security Cache Timeout: 600 seconds

Figure 117. Enabling security: Global

The *Security Cache Timeout* value is how long the server will retain security information regarding users. We will leave this on the default of 600 seconds.

We are going to use the default settings for all the other values in this task wizard as well, but it is worth looking through them to understand what is happening.

Figure 118 shows the other three pages for global security: Application Defaults, Authentication Mechanism, and User Registry.



Realm
* Name: almaden.ibm.com
Challenge Type
None
[✓ Basic (User ID and Password)]
Certificate
Default To Basic
Custom
* Login URL
* Relogin URL
Use SSL to connect client and Web serve

Authentication Mechanism
[✓ Local Operating System]
Lightweight Third Party Authentication (LTPA)
* Token Expiration (minutes): 30

Enable Single Sign On (SSO)
* Shared Name
* Domain
Limit to SSL connections only

Local Operating System
* Security Server ID: itso
* Security Server Password: ****
* - indicates a required field

Figure 118. Global security defaults

Application defaults

On the *Application Defaults* page, you specify the security realm that your enterprise applications will belong to. If so configured, a user is prompted only once for their identity information within a realm, no matter how many different resources they access.

You can also specify the default challenge type that your enterprise applications will use. This is how the Web server will ask the user to identify themselves. For the purposes of our book, basic HTTP authentication is fine. This will simply ask the user to provide a user ID and password.

You can also opt to use an SSL connection between the client and Web server, but this is a little excessive for our examples.

Authentication mechanism

On the *Authentication Mechanism* page, choose what system the user will be authenticated against. You can either use the local operating system's user registry, or you can connect to a Lightweight Third Party Authentication (LTPA) system, for example, IBM SecureWay Directory. To keep our example simple, we are going to use the Windows NT User Registry as our authentication mechanism.

User registry

The contents of the *User Registry* page varies depending on the selection made in the *Authentication Mechanism* page. The image shown in Figure 118 is a result of having chosen the local operating system as our authentication mechanism.

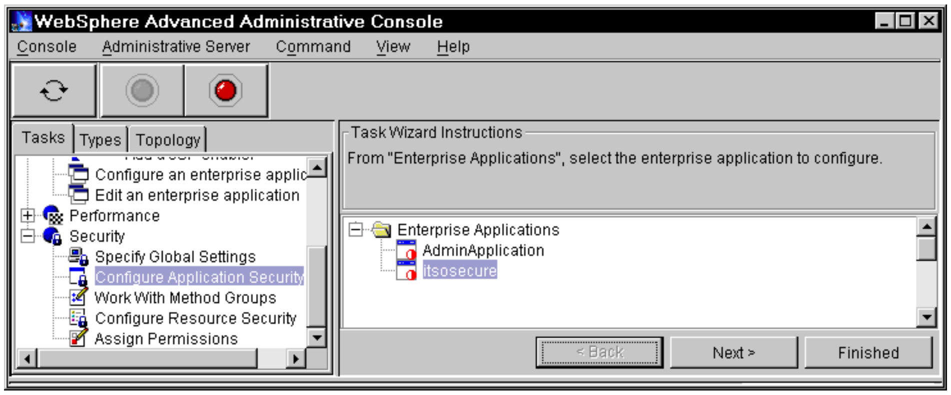
The user ID and password that the security server will use to run under should have been pre-filled in, based on information given during the install of WAS. Again, you can leave these fields as they are.

Click on *Finished* when you have reviewed all the settings, and you will be informed that your changes will not take effect until the Administration server is restarted. We will delay this step until we have finished configuring all of our security settings.

Configure enterprise application security

The next step is to configure the security at the individual enterprise application level. Even if you just want to use the default settings that you configured in “Specify global settings” on page 152, you still have to complete this step.

Start the task wizard called *Configure Application Security*. You are prompted to select the enterprise application that you want to configure (Figure 119).



From "Enterprise Applications", select the enterprise application to configure.

Figure 119. Choosing an enterprise application to secure

It is worth noting that, in addition to the enterprise applications you have created, there is also one named *AdminApplication*. This is used by WebSphere to provide security for the administrative console. A side effect of enabling global security is that the next time you start the administrative console, you are asked for a user ID and password (see “Restarting the administration server” on page 160).

Select *itsosecure*, and click on *Next*. This displays the same dialog as for global application defaults (Figure 118 on page 154). Here you can override many of the default values that you set up as global settings. However, we will leave all the values set to the defaults. Click on *Finished*, and the enterprise application is configured for security, although individual resources still have to be configured.

Method groups

Method groups are categories of methods, grouped for the purpose of assigning permissions to the method group as a whole, instead of having to assign permissions at an individual level.

WAS provides six default groups, and you can create your own method groups through the *Work with Method Groups* task wizard. For the purposes of our example, we will just use the default groups that have already been created.

Default method groups

❑ReadMethods

- GET and POST methods of Web resources
- Enterprise bean methods that have the prefix *get*

❑WriteMethods

- PUT methods of Web resources
- Enterprise bean methods that have the prefix *set*

❑RemoveMethods

- DELETE methods of Web resources
- REMOVE methods of an enterprise bean Home

❑CreateMethods

- CREATE methods of an enterprise bean *Home*

❑FinderMethods

- FIND methods of an enterprise bean *Home*

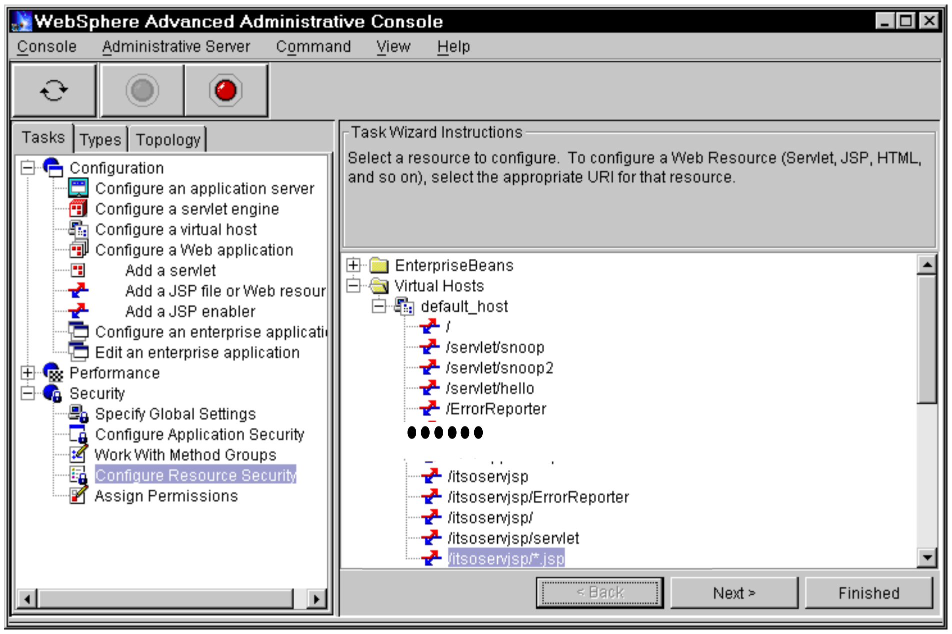
❑ExecuteMethods

- Methods that do not fit in the other default categories.

You do not have to run this wizard if you only use the default method groups.

Configuring the resource security

Now we need to actually place some security on individual resources. Start the *Configure Resource Security* task wizard. Here you can select the Web resource that you want to protect (Figure 120).



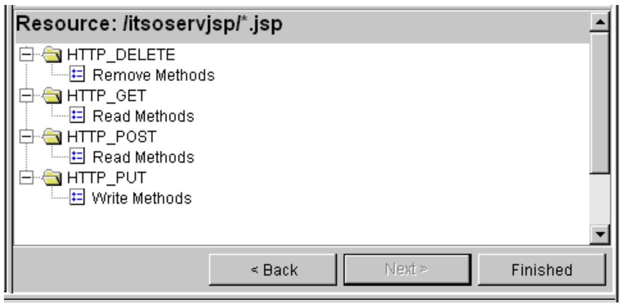
Select a resource to configure. To configure a Web Resource (Servlet, JSP, HTML, and so on), select the appropriate URI for that resource.

Figure 120. Selecting a resource to configure for security

Expand the virtual hosts (default_host) to see the list of resources. Select one of the resources, for example, */itsoservjsp/*.jsp* (meaning all JSP files of our Web application) and click on *Next*.

You are prompted if you wish to use the default method groups. Because we did not create any of our own method groups, select *Yes*. On the following page you can see that the methods of the selected resource have been associated with their appropriate groups (Figure 121).

Click on *Finished*, and resource security for the selected resource is configured.


Resource: /itsoservjsp/*.jsp

- HTTP_DELETE
  - Remove Methods
- HTTP_GET
  - Read Methods
- HTTP_POST
  - Read Methods
- HTTP_PUT
  - Write Methods

Figure 121. Viewing methods associated with method groups

You have to repeat this step for all the resources that you want to be secure. All resources that you do not configure are left open and not secure. As a minimum, you should secure JSPs (/itsoservjsp/*.jsp) and your servlets (/itsoservjsp/simple), or you can secure all the resources that start with /itsoservjsp. Do not secure /itsoservjsp/ErrorReporter, otherwise security errors cannot be reported properly.

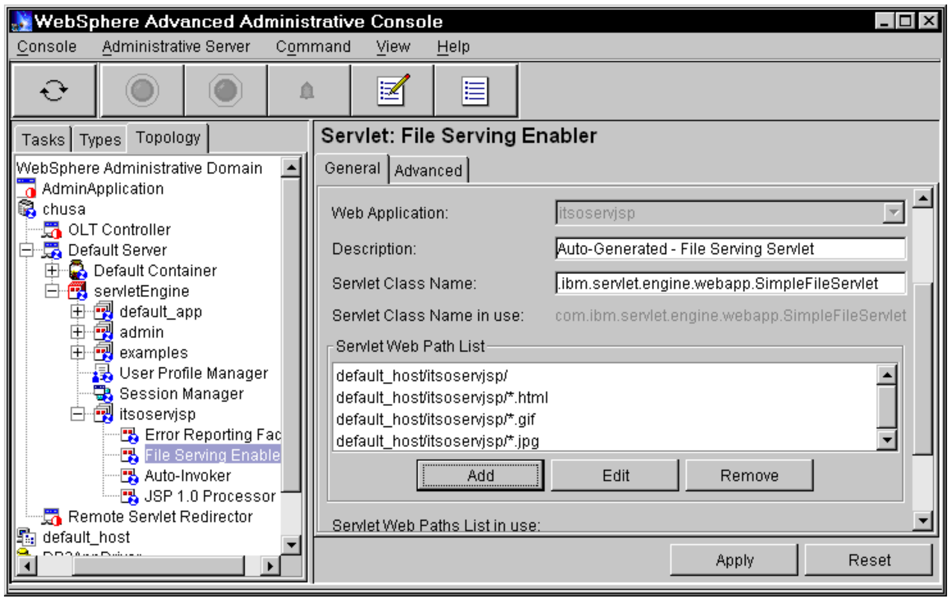
Configuring the File Serving Enabler servlet

If you want to add security to all normal files (for example, .html and .gif files) that are served by WAS, then you have to configure security for the *File Serving Enabler* servlet resource. The File Serving Enabler servlet is used by WAS to return Web files that are not served by the HTTP server.

In our example the default URI for the File Serving Enabler servlet resource is */itsoservjsp/*.

To make this work, you have to add all file types to the *Servlet Web Path List* for this servlet (Figure 122). Click *Apply* when all file types have been added. Otherwise, you may find yourself being able to load up a JSP, but an image it contains will be reported as not found.

Restart the Web application after adding file types to the Web path. When you select the File Serving Enabler servlet again, you should see the file types in the bottom pane.



Servlet: File Serving Enabler

Web Application: itsoservjsp
Description: Auto-Generated - File Serving Servlet
Servlet Class Name: .ibm.servlet.engine.webapp.SimpleFileServlet
Servlet Class Name in use: com.ibm.servlet.engine.webapp.SimpleFileServlet

Servlet Web Path List
default_host/itsoservjsp/
default_host/itsoservjsp/*.html
default_host/itsoservjsp/*.gif
default_host/itsoservjsp/*.jpg

Servlet Web Paths List in use:

Figure 122. File Serving Enabler servlet Web path list

Resource Security

Go back to the *Configure Resource Security* task wizard. The new resources that you added to the File Serving Enabler are now listed as well. Select each one and add them to the default method groups; when done, click *Finished*.

Assigning permissions

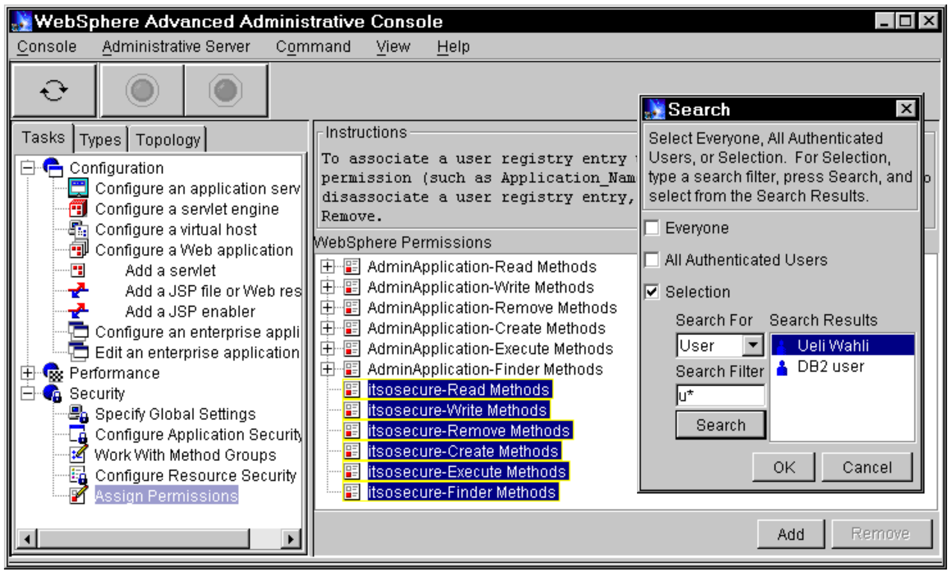
Once you have specified which resources in your enterprise applications are protected, you have to assign method groups to users to set up which user(s) are to have permission to see these resources.

Select the *Assign Permissions* task, and a list of all the enterprise application and method groups pairings is displayed. Select individual pairings or all the *itsosecure-Xxxx Methods*, and click on *Add* (Figure 123).

You are prompted to identify a user or user group that can access that application — method group pairing.

You can select *Everyone*, *All Authenticated Users* (by the operating system or LDAP), or *Selection*. For *Selection* you can use the search capability to search for the individual user that you want to give access to the resource pairing. You can select a number of users or even user groups, but for the purposes of our example, a single user is sufficient. When done with selecting users, click on *OK*.

One user is now allowed to access the selected application - method group pairings. The permissions configuration is now complete.



To associate a user registry entry
permission (such as Application_Nam
disassociate a user registry entry,
Remove.

AdminApplication-Read Methods
AdminApplication-Write Methods
AdminApplication-Remove Methods
AdminApplication-Create Methods
AdminApplication-Execute Methods
AdminApplication-Finder Methods
itsosecure-Read Methods
itsosecure-Write Methods
itsosecure-Remove Methods
itsosecure-Create Methods
itsosecure-Execute Methods
itsosecure-Finder Methods

Select Everyone, All Authenticated Users, or Selection. For Selection, type a search filter, press Search, and select from the Search Results.

[ ] Everyone
[ ] All Authenticated Users
[✓] Selection

Search For: User
Search Filter: u*
Search Results: Ueli Wahli, DB2 user

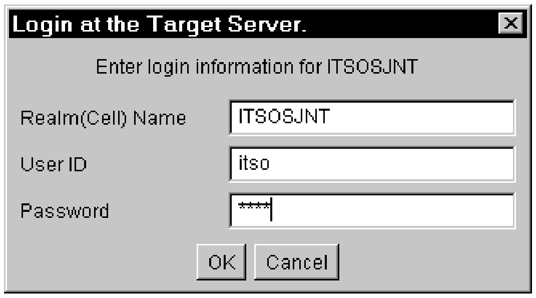
Figure 123. Assigning permissions to access method groups by users

Restarting the administration server

The final step before testing the newly-secured resources is to restart the administration server. You can do this by locating the node in the *Topology* view, and right-clicking on it and selecting *Restart*. Because the administrative console is running on the same node as the administration server, the console is automatically closed as well (after prompting you).

Check that the administration server has been restarted (by looking at the *Services* window in the *Control Panel*). When the server is up, start the administrative console. During start up, you are prompted for a user ID and password to access the console (see “Specify global settings” on page 152). Figure 124 shows the login with values that were configured for the security server.

When the administrative console appears, make sure that your application server and all its Web applications are started. You are now ready to test the security setup.


Login at the Target Server.

Enter login information for ITSOSJNT

Realm(Cell) Name: ITSOSJNT

User ID: itso

Password: ****

Figure 124. Login for the administration console

Testing your secure Web resources

After restarting the application server, test security from a browser by entering the URLs:

```
http://localhost/itsoservjsp/simple
http://localhost/itsoservjsp/very_simple.jsp
http://localhost/itsoservjsp/servlet/SnoopServlet
http://localhost/itsoservjsp/servlet/ServletEngineConfigDumper
```

You should be prompted once for user ID and password, and all the requests should complete.

Making further changes

If you wish to make any changes to the resource security and permissions at an enterprise application level, you can do so without having to restart the administration server. Simply make your changes, and then locate your application server in the Topology view, and restart it. When this is complete, your changes should be in effect.

XML configuration interface

WebSphere Application Server provides an XML interface that can be used to import and export definitions into the administrative database. This facility is a technology preview of the product.

The XML configuration utility is invoked as:

```
d:\WebSphere\AppServer\bin\xmlconfig -adminNodeName nodename
          -import input.xml
          -export output.xml -partial select.xml
```

The *nodename* is required and identifies the node for which the operation is performed.

For import, the *input* file contains the changes for the configuration.

For export, the *output* file contains the result. The -partial specification is optional and can be used to select what part of the configuration should be exported (default is the whole node).

Exporting configuration data

To export a complete or partial configuration for the *chusa* node, run:

```
xmlconfig -export chusa.xml -adminNodeName chusa
xmlconfig -export export.xml -partial select.xml -adminNodeName chusa
```

The *select.xml* file specifies what part of the system to export. For example, to export the *itsoservjsp* Web application, the select file would be:

```
<websphere-sa-config>
  <node name="chusa" action="locate">
    <application-server name="Default Server" action="locate">
      <servlet-engine name="servletEngine" action="locate">
        <web-application name="itsoservjsp" action="export">
        </web-application>
      </servlet-engine>
    </application-server>
  </node>
</websphere-sa-config>
```

To export a JDBC driver and a data source, the select file would be:

```
<websphere-sa-config>
   <jdbc-driver name="DB2AppDriver" action="export"> </jdbc-driver>
   <data-source name="sampledb" action="export"> </data-source>
</websphere-sa-config>
```

Importing configuration data

To import a definition, run:

```
xmlconfig -import input.xml -adminNodeName chusa
```

The *input.xml* file contains the definitions that have to be added or updated. For example, to define a JDBC driver and a data source, the input file would contain these statements:

```
<websphere-sa-config>
  <jdbc-driver name="DB2AppDriver" action="update">
    <implementation-class>COM.ibm.db2.jdbc.app.DB2Driver
    </implementation-class>
    <url-prefix>jdbc:db2</url-prefix>
    <jta-enabled>false</jta-enabled>
    <install-info>
        <node-name>fundy</node-name>
        <jdbc-zipfile-location>D:\SQLLIB\java\db2java.zip
        </jdbc-zipfile-location>
    </install-info>
  </jdbc-driver>
  <data-source name="sampledb" action="update">
    <database-name>sample</database-name>
    <jdbc-driver-name>DB2AppDriver</jdbc-driver-name>
    <minimum-pool-size>1</minimum-pool-size>
    <maximum-pool-size>30</maximum-pool-size>
    <connection-timeout>300</connection-timeout>
    <idle-timeout>1800</idle-timeout>
    <orphan-timeout>1800</orphan-timeout>
  </data-source>
</websphere-sa-config>
```

Note: Export of a JDBC driver generates an incomplete file in WebSphere Version 3.02; the install information is missing.

Examples

A number of import and export examples are provided in the sample code in the *wasxml* directory. See Appendix C, “Using the additional material” on page 417 for more information.

A number of examples for importing of definitions are provided in Part 2 of this book. Refer to “Tailor the XML files” on page 365 for examples.

User profiling

The need to manage user profiles within Web applications is becoming very common. Handily, WAS comes ready-built with some helper classes for managing your user profiles.

Built-in WAS functionality enables you to create a database table that will store user data, with common columns such as name and title provided. This table maps to the *com.ibm.websphere.userprofile.UserProfile* class. WAS also contains a *com.ibm.websphere.userprofile.UserProfileManager* class which allows you to perform the following tasks:

❑Creation and deletion of user profiles

❑Getting and updating (cached and immediate) from/to the database

❑Getting a user profile for read-only tasks

❑Queries on database columns

This should be enough to provide a solid foundation for building user profiling capabilities into your application, although you can easily extend the user profile for anything else you require.

For an example of implementing and extending user profiling in WAS Version 3, refer to *“The XML Files: Using XML and XSL with IBM WebSphere 3.0”*, SG24-5479.

Troubleshooting

Although you will carry out the majority of your development in the VisualAge for Java and WebSphere Studio environment described in earlier chapters, there will be occasions when you will need to perform some troubleshooting activities in WAS.

WAS provides two levels of tracing support:

❑Tracing within WAS

❑Tracing your application components using the Object Level Trace (OLT) and debugging tool.

We briefly discuss how to enable tracing within WAS, but will not cover using the OLT, as it is outside the scope of this book. For more detailed information on all aspects of tracing, please refer to the documentation that comes with the WAS product.

Tracing within WAS

Within WAS there are three different ways of collecting troubleshooting information:

- ❑Messages
- ❑Logs
- ❑Traces

Messages

These provide a high-level view of important events, such as successful completions and fatal errors, as your code runs on the Application Server product.

Console message pane

Messages are visible in the Console Message pane of the Administrative Console (see Figure 91 on page 127). If you require more detailed information regarding a message, you view it in the Serious Event Viewer.

Serious event viewer

This tool can be started from the menu option *Console* -> *Trace* -> *Serious Events*. Using this tool, you can show combinations of audit, fatal, terminate, and warning message events.

Logs

The WAS log files are kept within the `d:\WebSphere\AppServer\logs` directory. Here, you can find the standard error and standard output files for each application server that is running. In most setups, this will include the files: *default_server_stderr.log* and *default_server_stdout.log*. Servlets and JSPs will place the *System.out.println* output into these files.

Traces

These are collections of data from trace statements placed throughout the WebSphere product code or from any trace statements you may have added to your application code.

To enable tracing, select the menu option *Console* -> *Trace* -> *Enabled*. Obviously, enabling tracing will impact performance, and so it should be used sparingly.

Trace settings

It is possible to specify the trace settings for the administrative server and the application servers. You can specify which components the data is collected from, what type of data is collected, and where this data is placed (which file or which stream).

Monitoring resources

As well as tracing through the code, WAS provides a method of tracking resources running in your application server using the Resource Analyzer.

The Resource Analyzer can be started from under the Performance heading within the Tasks view. From here you can track different statistics for a number of resources, such as servlets, sessions, and data pools. You are then able to take this data and plot it in various combinations onto different types of charts, including graphs and pie charts.

For more detailed information on this topic and how to use the Resource Analyzer, refer to the documentation that comes with the WAS product.

Reference information

For more information about WebSphere Version 3. refer to the redbook *"WebSphere Application Servers: Standard and Advanced Editions"*, SG24-5460.

7 Development and testing with VisualAge for Java

In this chapter we discuss the VisualAge for Java environment for developing and testing Java servlets and JavaServer Pages (JSP).

We introduce the VisualAge for Java development environment for the development of servlets. We then discuss how to build, test, and run servlets in the WebSphere Test Environment under VisualAge for Java, and the use of the JSP Execution Monitor for the testing of JSPs. We show how to test complete Web applications, which include interacting application resources, such as servlets and JSPs; and passive application resources, such as static HTML files.

We will not discuss the Servlet Builder capabilities of the VisualAge for Java product. For information about the Servlet Builder, please refer to the IBM Redbook, "*VisualAge for Java Enterprise - Data Access Beans - Servlets - CICS Connector*", SG24-5265.

VisualAge for Java overview

The VisualAge for Java application development environment is shown in Figure 125.

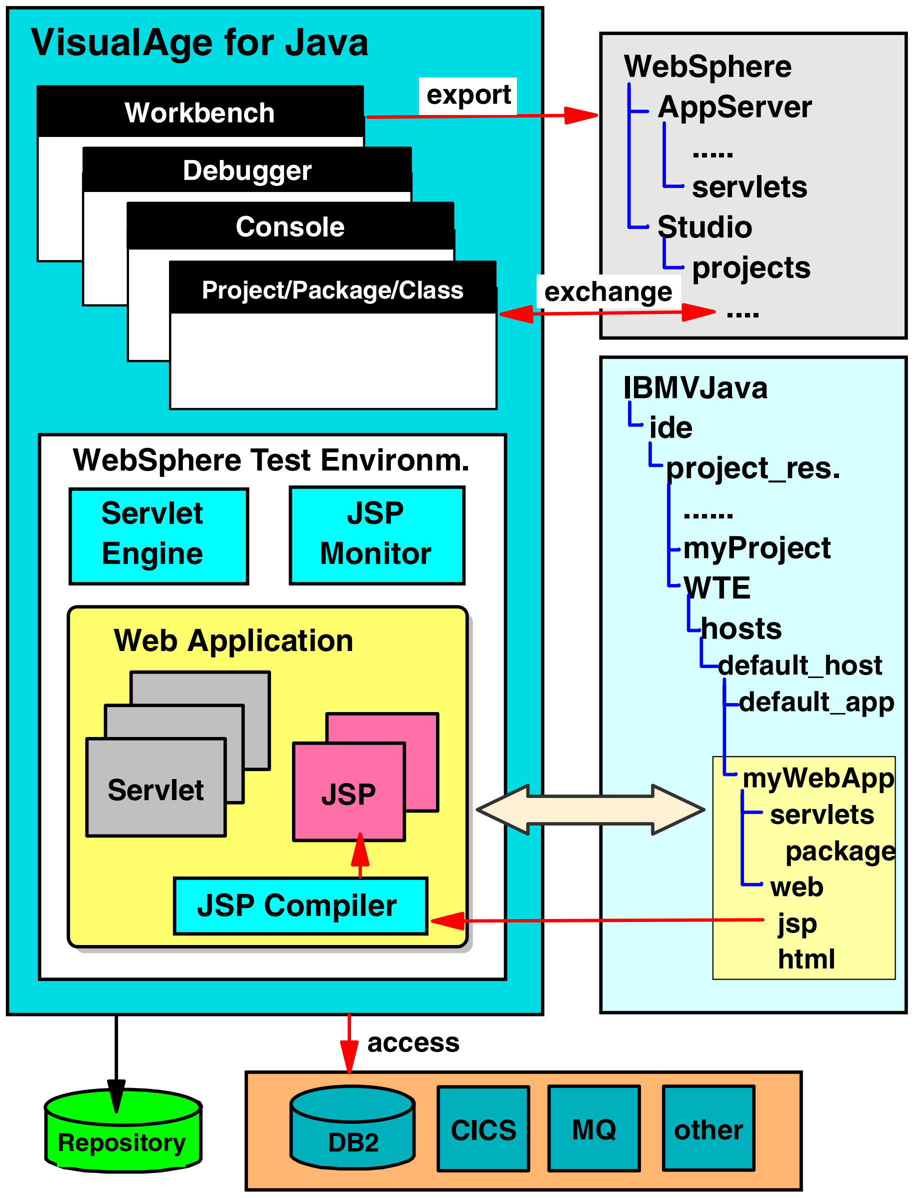


Figure 125. VisualAge for Java application development environment

Here is a short description of the major components of VisualAge for Java:

- ❑VisualAge for Java provides many windows to work with Java code:
  - The Workbench is the main window. It shows all the code that one developer works with.
  - The Debugger window is used to step through Java source code while debugging an application (or servlet).
  - The Console window shows output that normally would go to the system console.
  - The Browser windows for projects, packages, and classes provide views into smaller subsets of the code.
- ❑VisualAge for Java incorporates the WebSphere Test Environment that enables the testing of complete Web interactions involving HTML files, servlets, JSPs, and JavaBeans.
  - A servlet engine runs the servlet code.
  - HTML files, servlets, and JSPs can be grouped into Web applications to mirror the support that is available in WebSphere Application Server.
  - A JSP compiler compiles JSPs on first usage into servlets.
- ❑All the code of multiple developers is stored in a central repository and can be versioned.
- ❑The right side shows the directory structure. The most important directory for Web development is the *IBM WebSphere Test Environment* (abbreviated WTE) directory in the project_resources.

  Each Web application has its own subdirectories for servlets and Web resources (HTML and JSP).
- ❑Source and class files can be exported from VisualAge for Java into appropriate directories of WebSphere Application Server or WebSphere Studio.
- ❑WebSphere Studio provides a facility to interact directly with VisualAge for Java to exchange source and class files.
- ❑The code in Web applications can access enterprise resources, such as relational databases (DB2 and others), CICS, MQSeries, IMS, SAP, and others. This is normally done using the Common Connector Framework and connection pools.

Application development with VisualAge for Java

VisualAge for Java is a complete, integrated environment for creating Java applications. We first familiarize you with the VisualAge for Java development environment essentials, so that you have a solid background before beginning your servlet development.

This is a summary of the information that can be found in the product documentation, and is structured to get you on the fast-track to developing, testing, and debugging your servlets.

For additional information about the VisualAge for Java product, refer to `http://www.ibm.com/software/ad/vajava/`.

Rapid application development (RAD)

You can use VisualAge for Java's visual programming features to quickly develop Java applets and applications, using the Visual Composition Editor. Although we do not use the Visual Composition Editor in this chapter, we mention it here because it is a key component of the VisualAge for Java environment.

SmartGuides

In addition to its visual programming features, VisualAge for Java gives you SmartGuides (Wizards) to lead you quickly through many development tasks, including:

❑Creating new applets

❑Creating new program elements, such as:

- *Project:* The top-level program element in VisualAge for Java. A project contains packages. Projects are for organizational purposes, for example, versioning and deployment.
- *Package:* The Java language construct. Packages contains classes and interfaces, also called types.
- *Class:* The Java language construct. Classes contain methods and fields.
- *Interface:* The Java language construct. Interfaces contain methods and fields. The fields in interfaces must be static final fields.
- *Method:* The Java language construct.

❑The ability to visually create and manage JavaBeans and Enterprise JavaBeans.

- Importing and exporting of code from the file system, for deployment, or for integration with other tools, such as WebSphere Studio and WebSphere Application Server.

Create industrial-strength Java applications

VisualAge for Java gives you the programming tools that you need to develop industrial-strength code. Specifically, you can:

- Use the completely integrated visual debugger to examine and update code while it is running
- Build, modify, and use JavaBeans and BeanInfo classes
- Browse your code at the level of project, package, class, or method
- Package code into projects and packages
- Share a common repository of code among team members
- Instantly see all program errors across all projects and packages

Maintain multiple editions of programs

VisualAge for Java has a sophisticated code management system that makes it easy for you to maintain multiple editions of programs. When you want to capture the state of your code at any point, you can version an edition. An edition is a specific cut of a program element. This marks the particular edition as read-only, and allows you to give it a version identification.

VisualAge also provides integration with other external Source Code Management Systems (SCMS), such as ClearCase, Microsoft Visual SourceSafe, and PVCS.

VisualAge for Java components

In this section we describe the key components of the VisualAge for Java development environment:

Development with a repository

Within the VisualAge for Java environment, you do not manipulate Java code files directly. Instead, VisualAge for Java manages your code in a database of structured objects, called a repository. VisualAge for Java shows code to you as a hierarchy of program elements:

- *Project:* This is the highest organizational level within VisualAge for Java, also referred to as the application level, and it contains packages.

- ❑*Package:* Packages in VisualAge for Java are basically Java packages, a related grouping of classes in an application.
- ❑*Class* or *interface:* These are the individual source code elements, also called *types*.
- ❑*Method:* This is an individual method of a type.

Because you are manipulating program elements rather than files, you can concentrate on the logical organization of the code without having to worry about file names or directory structures.

The workspace and the repository

All activity in VisualAge for Java is organized around a single workspace, which contains the code for the Java programs that you are currently working on. The workspace also contains all the packages, classes, and interfaces that are found in the standard Java class libraries and other class libraries that you may need. When you exit from VisualAge for Java, the workspace is stored as a file.

While you work on code in the workspace, the code is automatically stored in a repository. In addition to storing all the code that is in the workspace, the repository contains other packages that you can add to the workspace if you have to use them.

In Chapter 3, “Product installation”, Figure 12 on page 25, we added two features that we need for servlet development to the workspace.

Importing and exporting code

You can easily move your code between your file system and VisualAge for Java. If you want to bring existing Java code into VisualAge for Java, you use the Import SmartGuide to specify the files (or whole directory structures) that you want to bring in. VisualAge for Java compiles your code, indicates if there are any errors, and adds the appropriate program elements to the workspace.

When you want to run your program outside of VisualAge for Java, you can export it using the Export SmartGuide. VisualAge for Java creates a Java source (*.java) file or compiled (*.class) file for each class that you export.

All the source code used in this redbook can be found at the ITSO Web site. The detailed instructions for importing this code into the VisualAge for Java environment can be found in Appendix C, “Using the additional material” on page 417.

The Workbench

VisualAge for Java gives you a variety of ways to examine and manipulate your code using different windows. The primary window you use in VisualAge for Java is called the Workbench. This window displays all the program elements in the workspace. It is important to make a clear distinction between the Workbench and the workspace. The Workbench is a window in the VisualAge for Java user interface. It displays the program elements that are in the user's workspace.

Pages (tabs) in the Workbench

Each page gives you a specific viewpoint on the application and code in the workspace.

Projects page

The Projects page displays all the projects in the workspace. You can expand projects to see the contained program elements (Figure 126). For this book we created a project named *ITSO Servlet JSP Redbook*.

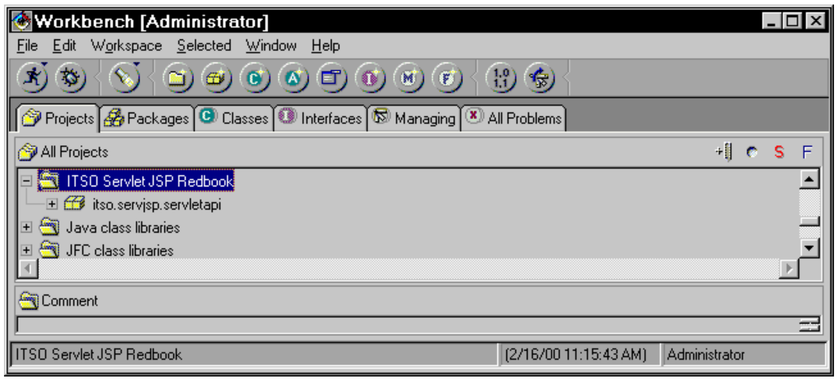
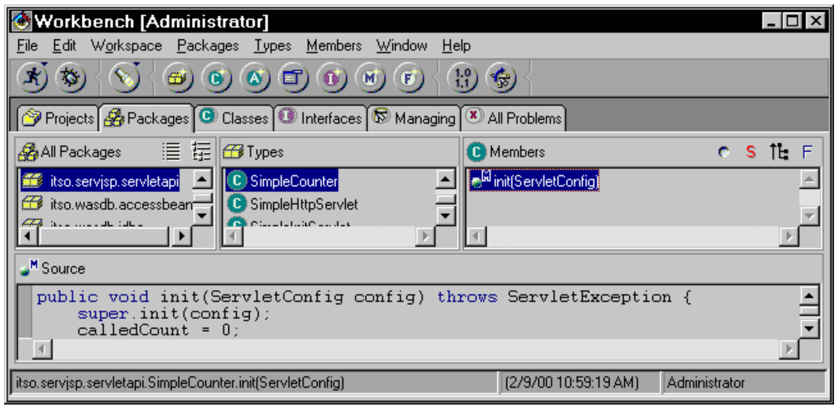


Figure 126. Projects page in Workbench

Packages page

The Packages page displays all the packages in the workspace. You can expand packages to see the contained program elements (Figure 127).


Workbench [Administrator]
File Edit Workspace Packages Types Members Window Help
Projects Packages Classes Interfaces Managing All Problems
All Packages
itso.servjsp.servletapi
itso.wasdb.accessbean
Types
SimpleCounter
SimpleHttpServlet
Members
init(ServletConfig)
Source
itso.servjsp.servletapi.SimpleCounter.init(ServletConfig)
(2/9/00 10:59:19 AM)
Administrator

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    calledCount = 0;
```

Figure 127. Packages page in Workbench

Classes page

The Classes page displays all the classes in the workspace in a hierarchy rooted at java.lang.Object. You have the choice of displaying the hierarchy as a list or as a graphical view. You can expand a class to see what classes inherit from it. Figure 128 shows some of the classes in the ITSO Servlet JSP Redbook project.

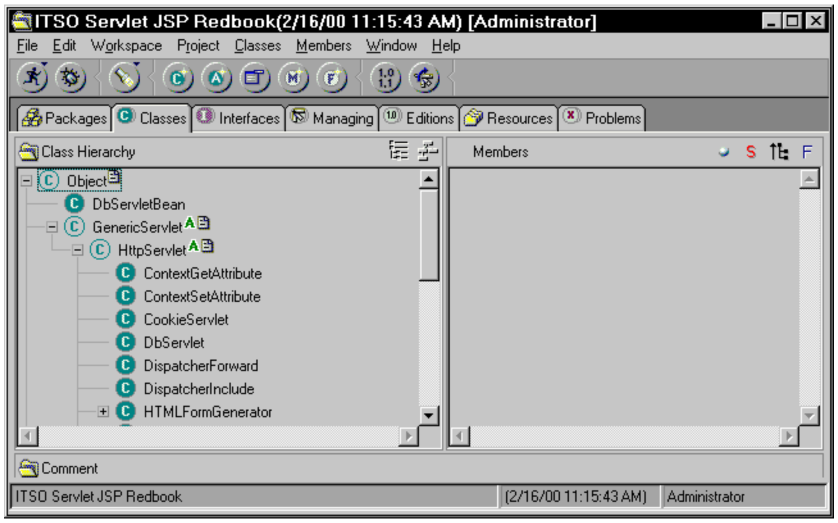
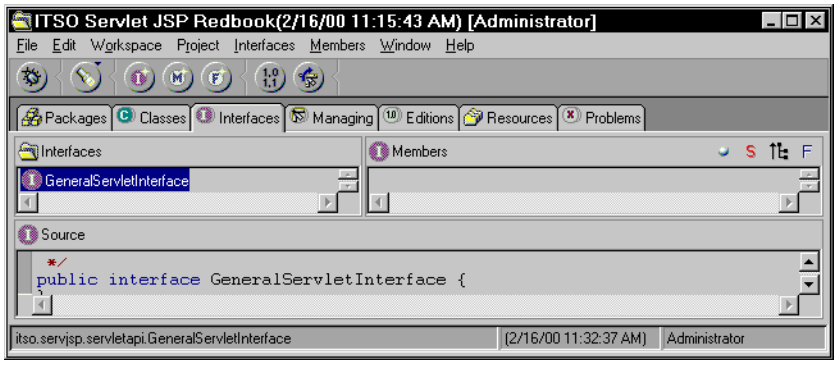


Figure 128. Classes page in Workbench

Interfaces page

The Interfaces page displays all the interfaces in the workspace. Figure 129 shows the interfaces in the ITSO Servlet JSP Redbook project.


ITSO Servlet JSP Redbook(2/16/00 11:15:43 AM) [Administrator]
File Edit Workspace Project Interfaces Members Window Help
Packages Classes Interfaces Managing Editions Resources Problems
Interfaces
GeneralServletInterface
Members
Source

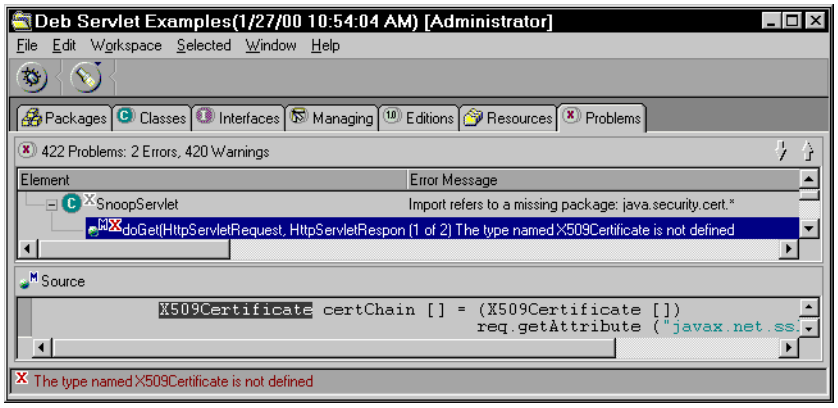
```
*/
public interface GeneralServletInterface {
```

itso.servjsp.servletapi.GeneralServletInterface (2/16/00 11:32:37 AM) Administrator

Figure 129. Interfaces page in Workbench

All Problems page

The All Problems page displays all the classes and methods in the workspace that have unresolved problems (Figure 130). When you save code, VisualAge for Java compiles it automatically.


Deb Servlet Examples(1/27/00 10:54:04 AM) [Administrator]
File Edit Workspace Selected Window Help
Packages Classes Interfaces Managing Editions Resources Problems
Source

422 Problems: 2 Errors, 420 Warnings

Element	Error Message
SnoopServlet	Import refers to a missing package: java.security.cert.*
doGet(HttpServletRequest, HttpServletRespon	(1 of 2) The type named X509Certificate is not defined

```
X509Certificate certChain [] = (X509Certificate [])
                                    req.getAttribute ("javax.net.ss.
```

The type named X509Certificate is not defined

Figure 130. Problems page in Workbench

Navigating in VisualAge for Java

VisualAge for Java gives you many ways to look at and manage your code. This section gives you a brief overview of primary windows in the VisualAge for Java environment, and tells you how to move from one window to another.

Moving between windows

Every window in VisualAge for Java has a *Window* menu. You can move between windows by selecting the window you want from this menu. Additionally, double-clicking on a program element may bring up the specific window browser associated with that element.

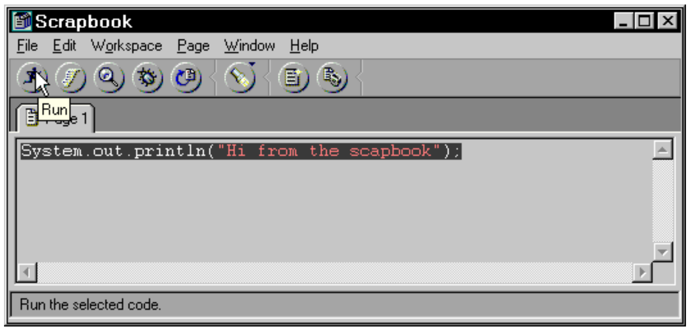
If the window you select is already open, it becomes the active window. If the window you want is not open, it is opened and becomes the active window. If you select *Switch To* in the *Window* menu, you can select from any of the windows that are currently open.

Windows you can open from the Window menu

Here is a summary of the windows that you can open from the Window menu.

Scrapbook

The Scrapbook window is a place to try out code (Figure 131). You can enter and run code fragments, without making them a part of any project, package, or class.

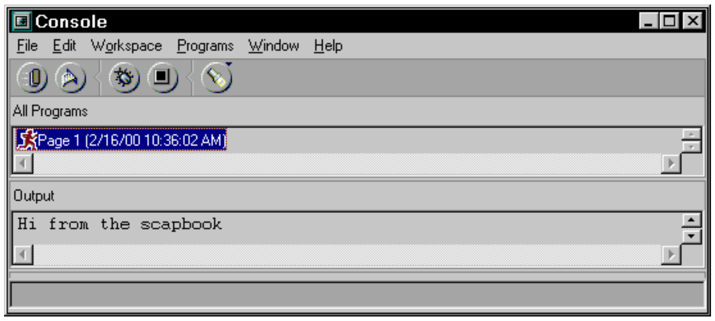


```
System.out.println("Hi from the scapbook");
```

Figure 131. Scrapbook window in VisualAge for Java

Console

The Console window displays standard out. It also gives you an area for entering input to standard in. If more than one thread is waiting for input from standard in, you can select which thread gets the input. Figure 132 shows the Console window (with the results from the scrapbook above).

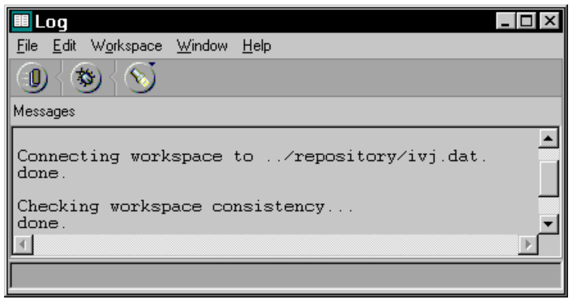

Console
File Edit Workspace Programs Window Help
All Programs
Page 1 (2/16/00 10:36:02 AM)
Output

```
Hi from the scapbook
```

Figure 132. Console window in VisualAge for Java

Log

The Log window displays messages and warnings from VisualAge for Java, for example, when a new edition is created from a version (Figure 133).

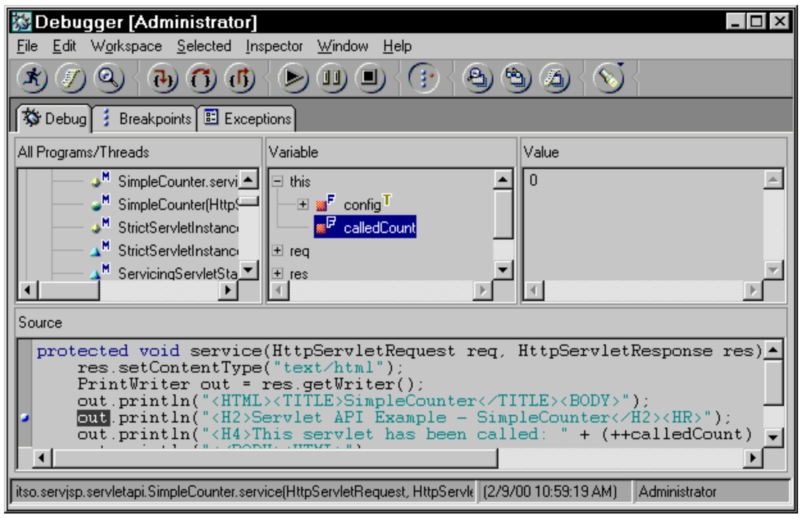


```
Connecting workspace to ../repository/ivj.dat.
done.

Checking workspace consistency...
done.
```

Figure 133. Log window in VisualAge for Java

Debugger

The Debugger window displays running threads and the contents of their runtime stacks (Figure 134). In the Debugger you can suspend and resume execution of threads; inspect and modify variable values; and set, remove, and configure breakpoints.



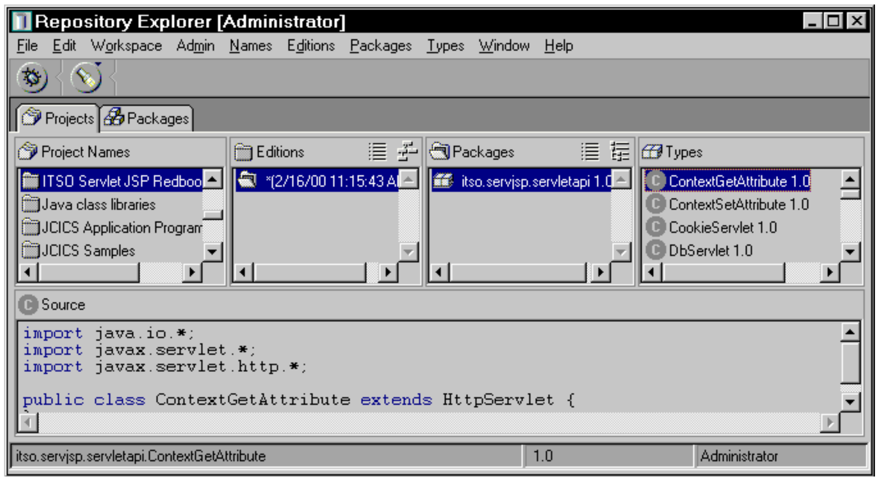
```
protected void service(HttpServletRequest req, HttpServletResponse res)
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>SimpleCounter</TITLE><BODY>");
    out.println("<H2>Servlet API Example - SimpleCounter</H2><HR>");
    out.println("<H4>This servlet has been called: " + (++calledCount)
```

```
itso.servjsp.servletapi.SimpleCounter.service(HttpServletRequest, HttpServle  (2/9/00 10:59:19 AM)  Administrator
```

Figure 134. Debugger window in VisualAge for Java

Repository Explorer

The Repository Explorer window displays all the editions of program elements in the repository (Figure 135). In the Workbench you can only find one edition of program elements that were loaded from the repository.



```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ContextGetAttribute extends HttpServlet {
```

```
itso.servjsp.servletapi.ContextGetAttribute   1.0   Administrator
```

Figure 135. Repository Explorer window in VisualAge for Java

Searching

The IDE gives you several choices for searching program elements. For example, in program element panes, if you press a letter key, VisualAge for Java selects the first displayed program elements that begins with that letter.

You have the following search options:

- ❑Searching with the Search dialog
- ❑Searching for references and declarations
- ❑Searching from the Workspace menu
- ❑Searching for a program element within a browser page

Browsing

VisualAge for Java gives you extensive facilities for browsing program elements. In the IDE, you browse a program element by opening it. There are many ways to open a program element in VisualAge for Java, but for now, here are two simple methods:

- ❑Select the program element, and select *Open* from the *Selected* menu or from the pop-up menu of the program element.
- ❑Select the appropriate browser in the Workspace menu, (Open Type Browser, Open Package Browser, Open Project Browser) for the program elements.

Project browser

The Project browser displays the following pages of one project: Packages, Classes, Interfaces, Editions, and Problems (Figure 136).

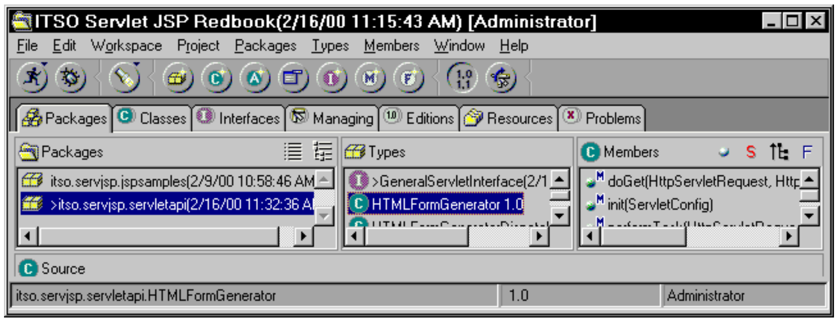

ITSO Servlet JSP Redbook(2/16/00 11:15:43 AM) [Administrator]
File Edit Workspace Project Packages Types Members Window Help
Packages Classes Interfaces Managing Editions Resources Problems
Packages
Types
Members
itso.servjsp.jspsamples(2/9/00 10:58:46 AM
>itso.servjsp.servletapi(2/16/00 11:32:36 A
>GeneralServletInterface(2/1
HTMLFormGenerator 1.0
doGet(HttpServletRequest, Http
init(ServletConfig)
Source
itso.servjsp.servletapi.HTMLFormGenerator
1.0
Administrator

Figure 136. Project browser

Package browser

The Package browser displays the following pages for one package: Classes, Interfaces, Editions, and Problems (Figure 137).

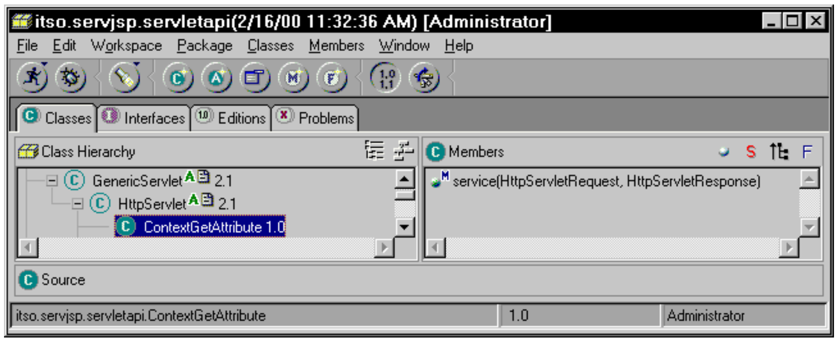
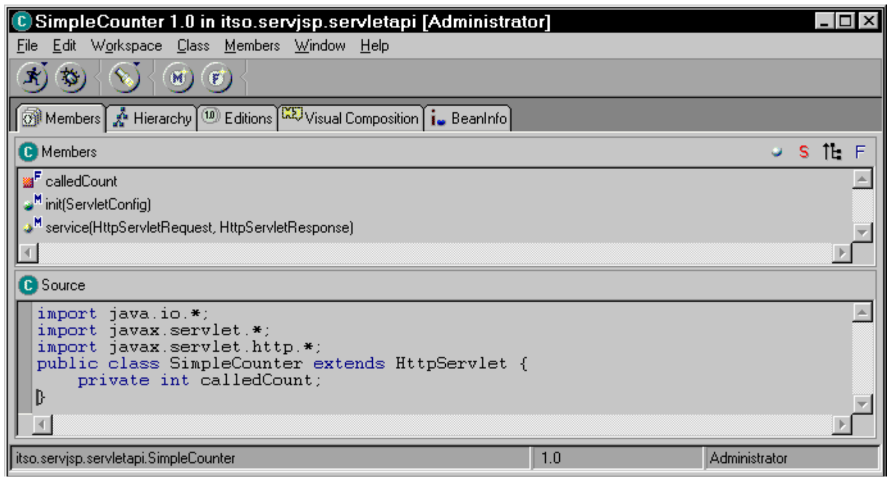

itso.servjsp.servletapi(2/16/00 11:32:36 AM) [Administrator]
File Edit Workspace Package Classes Members Window Help
Classes Interfaces Editions Problems
Class Hierarchy
GenericServlet 2.1
HttpServlet 2.1
ContextGetAttribute 1.0
Members
service(HttpServletRequest, HttpServletResponse)
Source
itso.servjsp.servletapi.ContextGetAttribute
1.0
Administrator

Figure 137. Package browser

Class browser

The Class browser displays the following pages for one class: Methods, Hierarchy, Editions, Visual Composition, and BeanInfo (Figure 138).

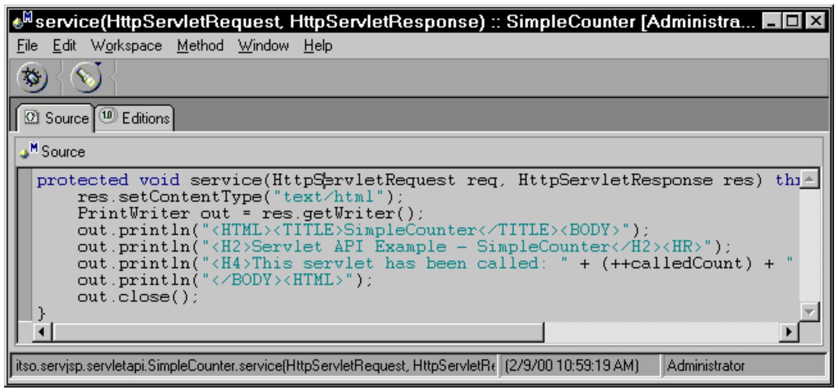

SimpleCounter 1.0 in itso.servjsp.servletapi [Administrator]
File Edit Workspace Class Members Window Help
Members Hierarchy Editions Visual Composition BeanInfo
Members
calledCount
init(ServletConfig)
service(HttpServletRequest, HttpServletResponse)
Source
itso.servjsp.servletapi.SimpleCounter
1.0
Administrator

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleCounter extends HttpServlet {
    private int calledCount;
```

Figure 138. Class browser

Method browser

When you open a method, you get a window with two pages: the Source page lists the source code, and the Editions page lists all the available editions (Figure 139).


service(HttpServletRequest, HttpServletResponse) :: SimpleCounter [Administra...
File Edit Workspace Method Window Help
Source
Editions
Source

```
protected void service(HttpServletRequest req, HttpServletResponse res) thı
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>SimpleCounter</TITLE><BODY>");
    out.println("<H2>Servlet API Example - SimpleCounter</H2><HR>");
    out.println("<H4>This servlet has been called: " + (++calledCount) + "
    out.println("</BODY><HTML>");
    out.close();
}
```

```
itso.servjsp.servletapi.SimpleCounter.service(HttpServletRequest, HttpServletRe  (2/9/00 10:59:19 AM)  Administrator
```

Figure 139. Method browser

Additional VisualAge for Java concepts

The following are a few additional items worth mentioning about the VisualAge for Java environment.

Adding features

We mentioned previously that you can use the SmartGuide to add features to the system as needed. You can access this SmartGuide through *File* -> *QuickStart* -> *Features* -> *Add Features*. The *F2* key is a fast path to the QuickStart menu.

Setting preferences

The VisualAge for Java IDE is a flexible work environment that you can adjust to meet your needs and preferences. You can change the way the IDE appears and functions by changing settings in the Options dialog.

To open the Options dialog, select *Options* from the *Window* menu, which appears on all IDE windows. The Options dialog contains settings for each of the customizable features. The settings are initially set to default values, but you can change the defaults to suit your work style or environment.

Java version

The VisualAge for Java product which we are running supports the Java API Version 1.1.7. There is another version of VisualAge for Java that supports Java 2.

VisualAge for Java IDE symbols

This section describes some of the symbols (icons) used in the VisualAge for Java environment. Use this mainly as a reference.

Hover help

When you move and hold the pointer over most symbols in the IDE, hover help and the status line present information about them, and the function that they perform. Figure 140 shows how the hover help displays when we move the mouse over the Search icon. From this page, we can also see symbols for creating projects, classes, applets, methods, and fields. We will discuss these more as we actually build our servlets.

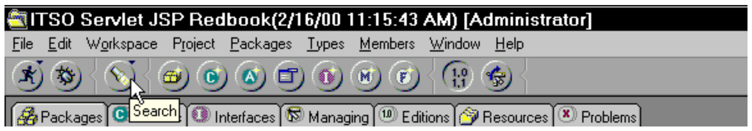

ITSO Servlet JSP Redbook(2/16/00 11:15:43 AM) [Administrator]
File Edit Workspace Project Packages Types Members Window Help
Search
Packages
Interfaces
Managing
Editions
Resources
Problems

Figure 140. Hover help actions

Other options from this page include the icons for creating packages, classes, applets, applications, methods, fields, opening and running the debugger, searching, viewing editions, and versioning.

The following symbols do not display this help, and may be used to display information about the programming elements.

Program elements

Figure 141 shows the symbols used to describe information at the program element level.

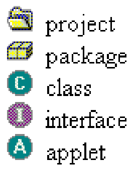
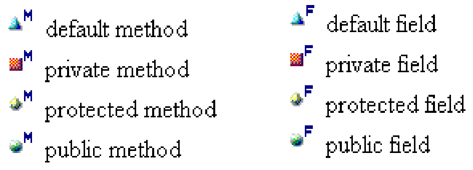


Figure 141. Program element symbols

Access modifiers for methods and fields

Figure 142 shows the symbols used to describe information about the program and its elements:

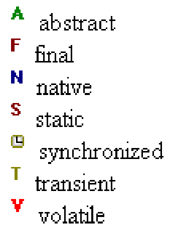


default method	default field
private method	private field
protected method	protected field
public method	public field

Figure 142. Program access modifiers

Other modifiers for classes, methods, and fields

Figure 143 shows the symbols used to describe additional information about the program and its elements:

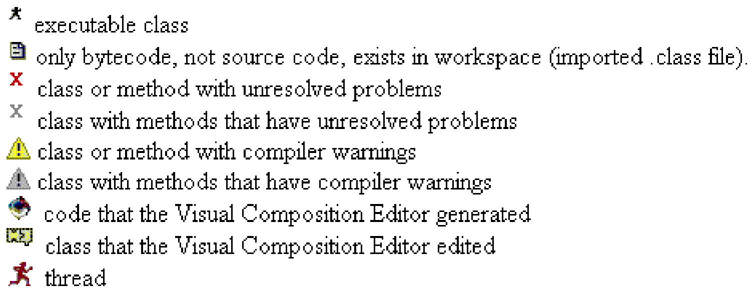


- abstract
- final
- native
- static
- synchronized
- transient
- volatile

Figure 143. Other modifiers

Other symbols

Figure 144 shows some other symbols in the VisualAge for Java environment:



- executable class
- only bytecode, not source code, exists in workspace (imported .class file).
- class or method with unresolved problems
- class with methods that have unresolved problems
- class or method with compiler warnings
- class with methods that have compiler warnings
- code that the Visual Composition Editor generated
- class that the Visual Composition Editor edited
- thread

Figure 144. Other symbols

Bookmarking elements

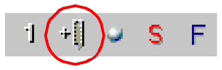
The Workbench lets you bookmark program elements, making it quick and easy to return to a frequently-used project, package, class, interface, or member within the Projects page. You can bookmark up to nine program elements.

To bookmark a program element:

❑In the Workbench, select the *Projects* page.

❑Select the program element you want to bookmark.

❑Select the bookmark button, located in the top right-hand corner of the *All Projects* pane. A bookmark number appears next to the bookmark button.



❑To see which bookmark relates to a particular program element, move the mouse pointer over it.

Code assist

Source panes, SmartGuides, and some other dialogs and browsers (for example, the Configure Breakpoints dialog) contain code assist, a tool to help you find the classes, methods, and fields you are looking for without having to refer to class library reference information. Code assist is accessed by typing *Ctrl+Spacebar*.

When you type *Ctrl+Spacebar*, classes, methods, parameters, and types that could be inserted in the code at the cursor are shown in a pop-up list, from which you can select one.

Servlet development

In this section we discuss how to develop servlets in the VisualAge for Java environment.

Rapid servlet development

VisualAge for Java gives you the ability to rapidly develop programs because of the Visual Composition editor and the built-in SmartGuides. VisualAge for Java gives you the ability to develop servlets visually as well, through the Servlet Builder interface. We have already mentioned that we will not cover the Servlet Builder in this chapter, but you might want to know why. In the following paragraphs, we provide some background about the servlet development process which might help explain this concept in more detail.

Model-View-Controller

We introduce the term Model-View-Controller (MVC) to describe a paradigm that has become popular in design, where we separate out the user interface in the view layer from the control layer, which manages the flow of the application, and the model layer, which handles our business logic and access to application resources.

We will describe this concept in more detail and other design techniques in Chapter 12, “Using Patterns for e-business to build the PDK” on page 347. This is a popular technique, because we can separate out our design components into different objects, making it a more modular, and object-oriented design.

Servlet-only applications

In servlet-only applications, the servlet is used as both the controller and the view. We can see this in our examples in Chapter 4, “Servlets” on page 41, where the servlets process our requests (control) *AND* produce the HTML response (view). In some cases, such as the JDBC servlet example, the servlet also acted as the model, because it referenced our data source.

Since we have tightly coupled the code for the controller and the view together in a single process, making changes to the HTML output will be difficult, because we have to modify source code, and recompile our program. One of the benefits of JSPs is that we can separate the view from the control.

Servlet-JSP applications

In servlet-JSP-based Web applications, discussed in Chapter 5, "JavaServer Pages" on page 95, the servlet can be used as the controller, and the JSP can be responsible for the view layer. (Note that the JSP can also be the controller, but this moves us away from a separation of view and controller.)

Servlet Builder

The Servlet Builder feature of VisualAge for Java enables us to build servlets visually, using JavaBean elements to visually design our servlet's HTML response page. Because we can do this HTML formatting in a JSP, and have a separate view layer to do so, we will not use the Servlet Builder for any of these examples. (We will be building JSPs visually in Chapter 8, "Development with WebSphere Studio" on page 227.) The Servlet Builder still has a role in development (although not implemented here) because it can be used to tie the servlet to other types of non-visual JavaBeans as well.

Even though we are not choosing to develop servlets using the Servlet Builder, VisualAge for Java is still a *very* rapid application development environment for servlets because of its many built-in features, including:

- ❑Built-in support for managing code versions
- ❑Easy Navigation
- ❑SmartGuides for developing everything from projects, packages, classes, and methods
- ❑Integrated visual debugging, which allows incremental code compilation while debugging a program.

The development process

The development process is more than just writing code. It is usually an incremental process that involves coding, testing, and debugging (and then some more coding). VisualAge for Java is uniquely suited for servlet development because it integrates all of these development tasks within a single environment.

We will use VisualAge for Java to develop and test our servlet examples. In a typical environment, you might develop your servlets in a tool such as VisualAge for Java, and then deploy them to run on a Web application server for testing. Additionally, your code could be shared with other development tools, such as WebSphere Studio (see Chapter 8, "Development with WebSphere Studio" on page 227).

VisualAge for Java provides not only a development environment in which to create and manage our servlets, but also a WebSphere Test Environment in which to run and test; as well as a debugger to interactively debug our servlets. We can also interactively run and debug JSPs in VisualAge using the JSP Execution Monitor. We discuss the WebSphere Test Environment and the JSP Execution Monitor in the next sections. Having integrated these development tasks in a single environment, VisualAge for Java is truly a rapid application development tool.

We will talk about each of these tasks later in this chapter, but for now, we will concentrate on how to *code* our servlets in VisualAge for Java. We will be creating our first servlet, *SimpleHttpServlet*, in VisualAge for Java.

Developing our first servlet

In walking through the steps below, we will be setting up our environment to allow us to create our first servlet, *SimpleHttpServlet*. We have already discussed the various windows in the VisualAge for Java environment. We will touch on each one when building our servlet.

Workbench

This is your IDE. It is the launching point for the other function specific browser windows. This is the window that you start on in the VisualAge tool. Start VisualAge, and go to the Workbench.

Workspace

This is your development environment. It contains references to all your source code, and it is customized and configured by you according to your specifications. The repository, which contains your workspace in addition to other VisualAge resources, is saved as a file `<IBMVJava>\ide\repository\ivj.dat`, where `<IBMVJava>` is your installation root. The workspace is stored as a separate file, `<IBMVJava>\ide\program\ide.icx`. All source code and resources become part of your workspace. You can see all the projects available to you in the workspace.

Projects

Projects are the highest level organizational structure within VisualAge for Java. All source code and application resources must belong to a project that enables a high level grouping of an application’s resources. You have to create a new project if you plan on running the servlet examples.

- ❑From the workbench, click on the *Project* (folder) icon to *Add New or Existing Project to Workspace* and enter *ITSO Servlet JSP Redbook* as the name of the project.

Packages

Packages in VisualAge for Java are like Java packages. It is a way to create groupings of related classes within an application. You will need to create a package to contain all of the servlet example source code that you create.

❑Select the *ITSO Servlet JSP Redbook* project.

❑Click on the *Package* icon to *Add New or Existing Package to Workspace*.

❑Create a new package named *itso.servjsp.servletapi*.

Note: If you have already imported the source code for this redbook, this package should already exist. If you still want to develop the source code from the ground up, we suggest you use a different project and package name (to guarantee uniqueness). However, we will use these naming conventions in this chapter.

Classes

What you actually create here are the Java source code programs which compile into Java class files. These classes should always be added to the *ITSO Servlet JSP Redbook* project and the *itso.servjsp.serlvetapi* package (unless this package already exists, and you have created your own).

To create a new Java source file:

❑Select the *ITSO Servlet JSP Redbook* project and the *itso.servjsp.servletapi* package in your workspace.

❑Click on *Create Class* icon to *Add New or Existing Class*. The project and package name should already be filled in and match the names above. Select the default options on this page.

❑Enter the class name as *SimpleHttpServlet*, and the superclass name as *javax.servlet.http.HttpServle*t. Select the default options on this page and click *Next*.

❑Add the following packages: java.io.*, javax.servlet.*, and javax.servlet.http.*.

❑Click *Finish*, and the skeleton class is generated.

Viewing and modifying the Java source code

The servlet which you just created above has the signature and methods of a servlet, but no implementation. You have to modify this new servlet to add some basic functionality. Figure 145 shows the source of the *service* method.

❑*Double-click* on the new class to open the Class browser.

❑Select the *service* method and edit the code to match Figure 145.

❑To save the code, hit *Ctrl-s*, or select another method (when a method object loses focus, you will force VisualAge for Java to re-compile the code if it has been changed).

```
protected void service(HttpServletRequest req, HttpServletResponse res)
                                    throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
   out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>");
   out.println("<H4>This is about as simple a servlet as it gets!</H4>");
   out.println("</BODY><HTML>");
   out.close();
}
```

Figure 145. SimpleHttpServlet: service method

Viewing the class declaration

To see the servlet class declaration, select the class. (If a method already has focus, use the *ctrl+mouse* combination, and reselect the class). You should see the Java source code declaration shown in Figure 146. One thing you do not see is the package declaration (`package itso.servjsp.servletapi;`). You do not programmatically have to specify this in the source code; VisualAge for Java adds it for you by default in the package where you create the class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
 * ITSO SimpleHttpServlet - Basic servlet example
 */
public class SimpleHttpServlet extends HttpServlet {

}
```

Figure 146. SimpleHttpServlet: class declaration

Viewing the complete source code

In VisualAge for Java, all of the individual properties and methods of your class file are stored as their own objects for easy manipulation within the IDE environment, and this is stored in the repository. You cannot see a full source code listing within VisualAge for Java; to do this you would have to export the source code from VisualAge to the file system. Because the VisualAge for Java environment itself can easily be navigated, you will find that you require the complete source very seldom, if ever. We will, however, discuss the details of importing and exporting in VisualAge later in this chapter.

Complete SimpleHttpServlet

Figure 147 shows the entire *SimpleHttpServlet* source code that you just created. This is the same *SimpleHttpServlet* that we introduced in Chapter 4, “Servlets” on page 41. The primary differences from the source code you see here, and what you see in the Servlets chapter, is that VisualAge for Java built a default constructor, and added a number of nice comments. We like the comments, but have eliminated them here to condense the code.

```
package itso.servjsp.servletapi;                <=== generated on export
```

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
 * ITSO SimpleHttpServlet - Basic servlet example
 */
public class SimpleHttpServlet extends HttpServlet {

}
```

```
/**
 * SimpleHttpServlet constructor comment.
 */
public SimpleHttpServlet() {
   super();
}
```

```
/**
 * service method comment.
 */
protected void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
   out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>");
   out.println("<H4>This is about as simple a servlet as it gets!</H4>");
   out.println("</BODY><HTML>");
   out.close();
}
```

Figure 147. SimpleHttpServlet: complete source code

WebSphere Test Environment

VisualAge for Java integrates much of the WebSphere Application Server Advanced runtime so that debugging servlets and JSPs (and EJBs) is possible in a highly integrated development environment. The WebSphere Test Environment (WTE) in VisualAge for Java actually encompasses the servlet and JSP runtime and test environments.

The WTE enables us to run our servlet examples in a controlled, simulated Web application server environment. Typically, one consequence of the servlet life-cycle is that you must normally stop and restart the Web application or reload the class file to apply your updated code changes. This can become tedious during development, when you are making lots of changes.

Fortunately, WTE offers a much more productive way to develop and test servlets (and JSPs) for WebSphere. When you change a method in the servlet, VisualAge for Java incrementally compiles only this modified method of the class, not the entire class, and hot-links it into the running program. This type of incremental compilation is an important productivity boost, because you do not have to stop and restart the WTE in programs that you are debugging to execute your updated code, and rebuild program state.

There are two different ways to configure and run the WebSphere Test Environment:

❑SERunner
❑ServletEngine

The first way, using *SERunner*, is described in this section, and is the primary method we use when talking about running the WebSphere Test Environment. We describe how to configure, run, and test our *SimpleHttpServlet* using the WTE within the VisualAge for Java tool.

The second way, using *ServletEngine*, is not documented (yet) in the VisualAge for Java help. With the ServletEngine we can configure the test environment to support multiple Web applications, to mirror what can be done in WebSphere. This gives us more flexibility and control over the testing environment, but is much harder to configure. We discuss this second method in more detail in “Configuring multiple Web applications” on page 215.

VisualAge for Java configuration for WebSphere

You need to ensure that the WebSphere Test Environment feature under VisualAge for Java have been successfully installed and configured. See “VisualAge for Java” on page 15 for detailed instructions on what features must be added to the VisualAge for Java workspace.

WebSphere Test Environment setup

There several steps that you have to perform to configure the WebSphere Test Environment for servlet development and testing:

❑In the Workbench, find the IBM WebSphere Test Environment project, and select the *com.ibm.servlet* package. Find the *SERunner* class.

❑Create a bookmark to this class by selecting the bookmark icon in the top right of the window. This step is optional, but will make it easier to find this class in the future.

❑Select *Run -> Check Class Path* in the context menu of the SERunner class.

❑Select the *Class Path* tab. Click the *Edit* button to edit the *Project Path*. Select the project *ITSO Servlet JSP Redbook*. Save this setting by selecting *OK*.

Note: You may have to come back here in the future to manipulate the class path settings for the application. One option (but unscientific and potentially unsafe with complex configurations) is to select all the projects and add them to the class path of the *SERunner* class.

Start the WebSphere Test Environment

Now we want to run the *SimpleHttpServlet* example in the WebSphere Test Environment.

There are a couple of ways to start the WebSphere Test Environment and launch the servlet. The WebSphere Test Environment starts a local WebSphere Application Server process (localhost, or 127.0.0.1), running on port 8080, by default.

Starting the SERunner class directly

To start the *SERunner* process directly, select the *SERunner* class, and select *Run -> Run Main* from the context menu. You can also click on the running man icon in the tool bar. Once started, you invoke the servlet from a Web browser by entering the appropriate URL of the servlet, for example:

```
http://localhost:8080/servlet/itso.servjsp.servlatapi.SimpleHttpServlet
```

Of course, this assumes you know the URL of the servlet, and we have not yet discussed this.

A more direct way to run your servlet is to use the Servlet Launcher.

Using the Servlet Launcher to launch the servlet

You can also start the *SERunner* class by using the Servlet Launcher capabilities. Select your servlet, *SimpleHttpServlet*, and *Tools* -> *Servlet Launcher* -> *Launch* from the context menu. If the *SERunner* is not yet running, it is started for you.

This method launches the servlet by starting a Web browser to invoke the servlet. The first time you launch the servlet, you are prompted for servlet parameters. This servlet does not require any parameters, so just click *OK* to continue.

Considerations when launching SERunner

If your servlet process requires any classes (jar files) that are not part of the VisualAge for Java workspace (for instance, external API or DB2 jars, such as `d:\SQLLIB\java\db2java.zip`), you have to either:

❑Import the jar or class directory into the VisualAge for Java workspace, and add this project to the *SERunner* class path, as described.

❑Select *Run* -> *Check Class Path* for the SERunner class and *Edit directories path*, then add the directory or jar file to the list.

❑Add the directory or jar file to the VisualAge for Java system class path in the Resources setting of the *Window* -> *Options* dialog.

Importing the classes or jar files increases the size of the workspace file. Additionally, unless you have the corresponding Java files, you will not be able to interactively debug your code.

This is why it may be desirable to use the second approach. This, however, forces you to always launch the SERunner before invoking a servlet. Otherwise, launching SERunner using the *Servlet Launcher* does not load the external classes.

The last approach is most often used for jar files that are required for many servlets and applications. For example, we suggest to add the D:\SQLLIB\java\db2java.zip to the workspace class path.

WebSphere Test Environment windows

If the SERunner class starts correctly, you will see a little pop-up window entitled WebSphere Test Environment (Figure 148). This is your indication that *SERunner* is running.

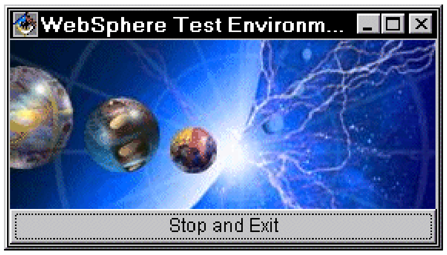


Figure 148. WebSphere Test Environment window

Stopping SERunner

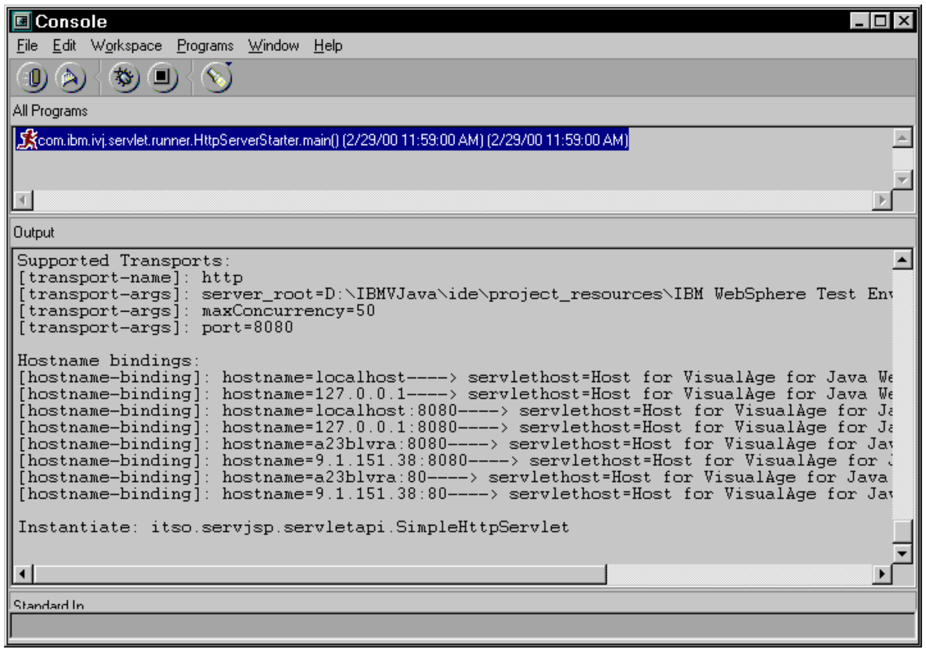
You can stop the *SERunner* process from the WebSphere Test Environment window. This will gracefully shut down the Web server and call the destroy methods for any loaded servlets. One of the nice things about this test environment, however, is that if you change your underlying class, you most likely do not have to restart *SERunner*, allowing for incremental development and debugging within the VisualAge for Java environment.

One situation where you may have to restart the *SERunner* is if you change the *init* method of the class. Because the *init* is processed only once within a servlet's life-cycle, changes to this method (such as the changing of initialization parameters), do not take effect until *SERunner* is restarted.

Console window

The VisualAge for Java Console window is also opened and displays the status of the *SERunner* process, and any servlets that you launch. The Console window basically displays the standard output and standard error of the Java program's execution. If there were problems starting up the environment, they would be display here. The messages that you see on successful start-up of the *SERunner* process are shown in Figure 149.

We mentioned that the status of any of our servlets is also displayed in this window. The simple servlet has started successfully if the line: `Instantiate: itso.servjsp.servletapi.SimpleCounter` appears in the Console window (Figure 149).



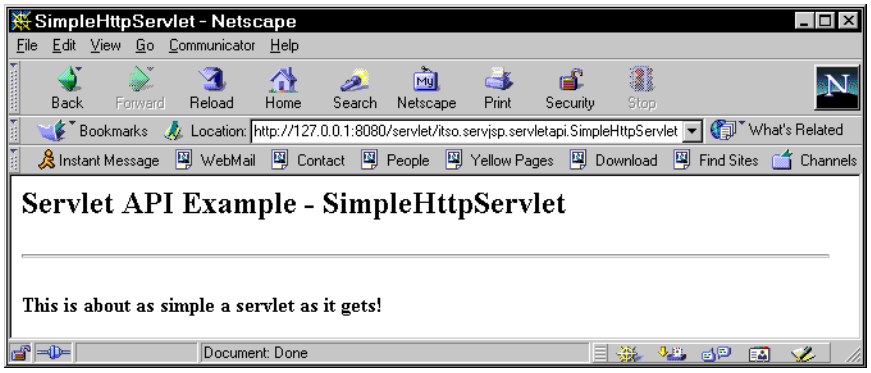
```
com.ibm.ivj.servlet.runner.HttpServerStarter.main() (2/29/00 11:59:00 AM) (2/29/00 11:59:00 AM)
```

```
Supported Transports:
[transport-name]: http
[transport-args]: server_root=D:\IBMVJava\ide\project_resources\IBM WebSphere Test Env
[transport-args]: maxConcurrency=50
[transport-args]: port=8080

Hostname bindings:
[hostname-binding]: hostname=localhost----> servlethost=Host for VisualAge for Java We
[hostname-binding]: hostname=127.0.0.1----> servlethost=Host for VisualAge for Java We
[hostname-binding]: hostname=localhost:8080----> servlethost=Host for VisualAge for Ja
[hostname-binding]: hostname=127.0.0.1:8080----> servlethost=Host for VisualAge for Ja
[hostname-binding]: hostname=a23blvra:8080----> servlethost=Host for VisualAge for Jav
[hostname-binding]: hostname=9.1.151.38:8080----> servlethost=Host for VisualAge for J
[hostname-binding]: hostname=a23blvra:80----> servlethost=Host for VisualAge for Java
[hostname-binding]: hostname=9.1.151.38:80----> servlethost=Host for VisualAge for Jav

Instantiate: itso.servjsp.servletapi.SimpleHttpServlet
```

Figure 149. SERunner Console status

Launching the browser

If you launched your servlet through the Servlet Launcher method, you should see the results of your servlet's execution displayed in your Web browser window. The results of the *SimpleHttpServlet* are shown in Figure 150.



Servlet API Example - SimpleHttpServlet

This is about as simple a servlet as it gets!

Figure 150. SimpleHttpServlet output

It is possible to invoke the servlet directly from the browser, without having to use the Servlet Launcher method. You would use this same URL, as long as the *SERunner* process is running. This may also be a desired approach, especially when you are testing servlets that interact with each other, or with other servlet resources. You do not want to be limited to having to invoke each servlet from the Servlet Launcher each time.

Web host path

Notice that the servlet is invoked with `http://127.0.0.1:8080` (or `http://localhost:8080`). There may be variations here based upon your TCP/IP settings, and any special configuration that you may do under WTE.

Servlet root path

The servlet Web path is `/servlet/itso.servjsp.servletapi.SimpleHttpServlet`. The */servlet/* path is the default for servlets running in the default application environment in the WebSphere Test Environment. This corresponds to the default_app Web application in the Web Application Server environment.

Fully qualified class name

The fully qualified class name, `itso.servjsp.servletapi.SimpleHttpServlet`, contains the package name, *itso.servjsp.servletapi*.

In our true WebSphere Application Server environment, we most likely would not invoke our servlets directly by their fully qualified name, because we would want to hide this implementation detail from the user. We do this by creating aliases for Web invocation. Because this is how servlets are invoked by default in the WTE, you have to keep this in mind when designing your programs, and use relative paths in your code when appropriate.

VisualAge for Java provides the facility to use multiple Web applications and servlet aliases. See “WebSphere Test Environment — multiple Web applications” on page 215 for more information.

What have we accomplished?

So far, we have been able to successfully create a Java servlet class in the VisualAge for Java development environment. We have modified the code, which caused VisualAge for Java to recompile it. We then set up the WebSphere Test Environment and launched the servlet using the *SERunner* process and the Servlet Launcher. We were able to see the results of the servlet’s execution in the Web browser.

You now have the basis for developing and running servlets in VisualAge for Java. The remainder of this chapter does not focus on how to create and develop more servlets. You should be able to repeat the steps above to accomplish that. Rather, we will focus on how to configure, run, and debug these servlets in VisualAge for Java.

Testing JSPs under WebSphere Test Environment

This section describes how to run JSPs under the VisualAge for Java environment, and how to have those JSPs interact with other servlets and/or JavaBeans.

VisualAge for Java configuration for JSPs

You have to make sure that the WebSphere test environment features under VisualAge for Java have been successfully installed and configured. See “VisualAge for Java” on page 15 for detailed instructions on the necessary features which must be added to the VisualAge for Java workspace.

Configuring the JSP version used by VisualAge for Java

Visual Age for Java version 3 supports both JSP .91 and JSP 1.0 versions, and defaults to JSP .91 for backward compatibility. To change the version of the JSP support used by the Visual Age Test Environment, perform these steps:

❑Open the configuration file of the default application:

```
d:IBMVJava\ide\project_resources\IBM WebSphere Test Environment
   \hosts\default_host\default_app\servlets\default_app.webapp
```

❑Find the JSP compiler servlet (Figure 151):

```
<servlet>
   <name>jsp</name><
   description>JSP support servlet</description>
   <code>com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet</code>
   <init-parameter>
   ...
```

Figure 151. default_app.webapp: JSP 0.91 configuration

❑Change the text for the <code> tag (Figure 152):

```
<servlet>
   <name>jsp</name><
   description>JSP support servlet</description>
   <code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code>
   <init-parameter>
   ...
```

Figure 152. default_app.webapp: JSP 1.0 configuration

Running our first JSP

JSPs in VisualAge for Java run in the same WebSphere Test Environment that servlets do. (After all, JSPs actually become servlets once they are page compiled.) Because we cannot create JSP files directly in the VisualAge tool (we will use WebSphere Studio to develop our JSPs), we have to make sure that *SERunner* can find the JSP files in the file system.

Location of JSP files

The default location for HTML and JSP files is specified in the file:

```
<IBMVJavaRoot>\ide\project_resources\IBM WebSphere Test Environment\
    SERunner.properties
```

in the line:

```
docRoot=D:\\IBMVJava\\ide\\project_resources\\IBM WebSphere Test Environment
\\hosts\\default_host\\default_app\\web
```

To qualify this path, you could create a subdirectory within this Web directory, for example, `\itsoservjsp`.

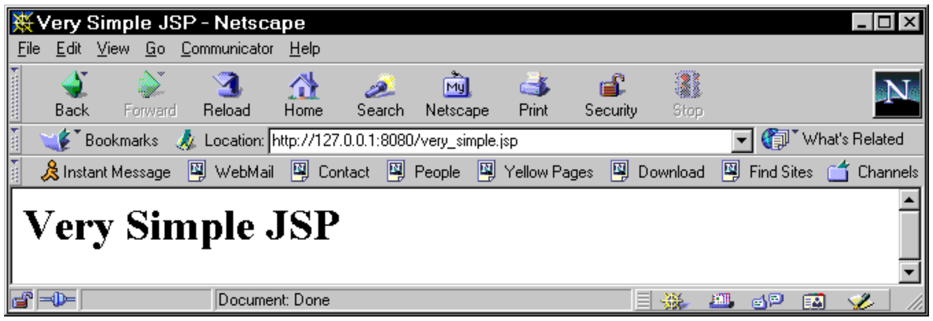
Running a simple JSP

VisualAge for Java ships with a couple of sample JSPs that we can use to test out our configuration, and see that JSPs have been enabled. Follow these steps to run the *very_ simple.jsp* example:

❑Start the *SERunne*r process and wait until it is ready.

❑Enter the following URL in a Web browser:
`http://127.0.0.1:8080/very_simple.jsp`

Figure 153 shows a successful JSP response.


Very Simple JSP - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://127.0.0.1:8080/very_simple.jsp
What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels
Document: Done

Very Simple JSP

Figure 153. Very simple JSP response

As the message indicates, this is a *VERY* simple JSP. In fact, the only tags it uses are HTML tags (Figure 154). It is essentially an HTML file saved with a .jsp extension.

```
<html><head><title>Very Simple JSP</title></head>
<body>
<h1>Very Simple JSP</h1>
</body>
</html>
```

Figure 154. Very simple JSP source

How do we know it ran as a JSP?

This file is still very much a JSP. This example does not have any advanced JSP tags, so how do we know that *SERunner* really ran it as a JSP and not as a regular HTML file? Its because of the .jsp file extension.

When *SERunne*r (and the WebSphere Application Server) receives a request for a .jsp file, it compiles this JSP into a servlet. This happens only the first time the JSP is requested; subsequent requests use the already compiled JSP. This JSP life-cycle is described more in the Chapter 5, “JavaServer Pages” on page 95.

So where is the compiled JSP stored? The JSP is translated into a servlet Java source file, and imported into VisualAge for Java (*JSP Page Compile Generated Code* project). The intermediate .java files, however, can be found in the file system, in the WebSphere Test Environment \temp\ directory:

```
<IBMVJava>\ide\project_resources\IBM WebSphere Test Environment
\temp\default_app\pagecompile\_very__simple_xjsp.java   <=== JSP 0.91
\temp\Jsp1.0\default_app\very_simple_jsp_0.java         <=== JSP 1.0
```

Creating and running a JSP

The *very_simple.jsp* file does not demonstrate much, except that the system *can* run JSPs. Our next step is to take one of the JSP examples from Chapter 5, “JavaServer Pages” on page 95, and run it within the VisualAge for Java WebSphere Test Environment.

We have not yet talked about the WebSphere Studio environment, which is a fully integrated Web site development environment that enables us to visually create JSPs (in addition to other application resources). So for now, you have to create this file by hand, using a standard editor of your choice.

Creating the JSP

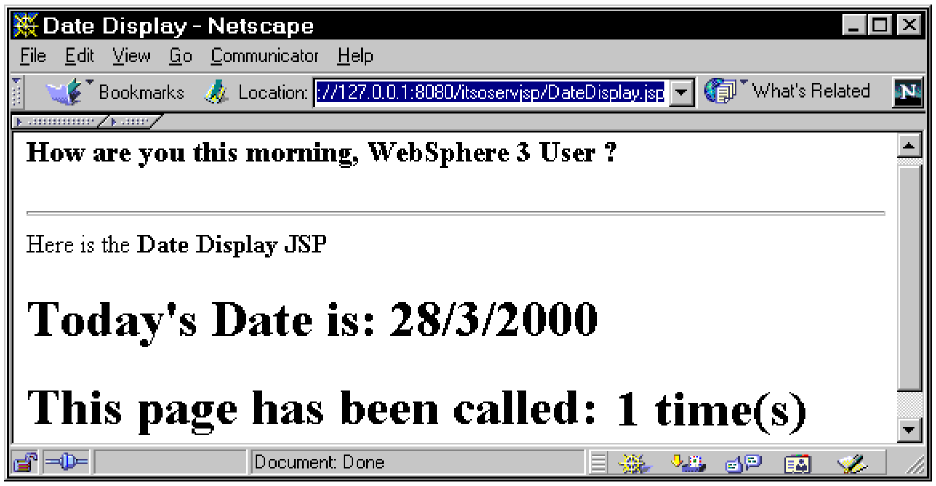
Create a new folder: `\...\hosts\default_host\default_app\web\`**`itsoservjsp`**. This is the root Web path that we will use for all of our JSP examples.

Create a new file in this directory, *DateDisplay.jsp*, and add the code from Figure 78 on page 106 from Chapter 5, “JavaServer Pages”. This JSP file contains many of the standard types of JSP tags, including directives, scriptlets, declarations, and accessing of implicit objects.

Run the JSP

Start the WebSphere Test Environment (*SERunner)* and enter the following browser command: `http://127.0.0.1:8080/itsoservjsp/DateDisplay.jsp`

Figure 155 shows the results of the JSP execution.


Date Display - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: ://127.0.0.1:8080/itsoservjsp/DateDisplay.jsp
What's Related
Document: Done

How are you this morning, WebSphere 3 User ?

Here is the **Date Display JSP**

Today's Date is: 28/3/2000

This page has been called: 1 time(s)

Figure 155. DateDisplay.jsp output

Debugging servlets and JSPs

VisualAge for Java includes an integrated visual debugger with a rich set of features. This section outlines some of these features, and describes how we can debug our servlets and JSPs within the VisualAge for Java environment.

Debugger basics

This section outlines the basics of the VisualAge for Java debugging environment.

Opening the debugger

You can open the debugger manually by selecting *Debug -> Debugger* from the *Window* menu. If a program is running, you can suspend its thread, and view its stack and variable values. Alternatively, the debugger will automatically open with the current thread suspended for the following reasons:

❑A breakpoint in the code is encountered.

❑A conditional breakpoint that evaluates to true is encountered

❑An exception is thrown and not caught

❑An exception selected in the Caught Exceptions dialog is thrown.

❑A breakpoint in an external class is encountered.

Setting breakpoints

When a program is running in the IDE and encounters a breakpoint, the running thread is suspended and the Debugger browser is opened so that you can work with the method stack and inspect variable values. In the IDE, you can set breakpoints in any text pane that is displaying source.

To set a breakpoint

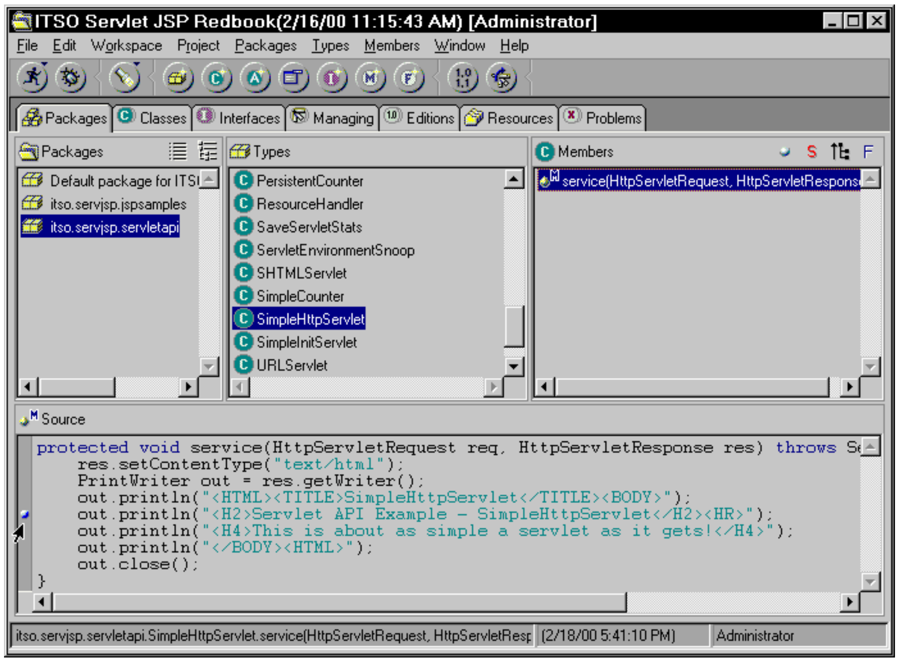
Find the code you want, and double-click in the left-margin of the source pane.

To remove a breakpoint

Find the breakpoint that is set in the code, and double-click on it in the left margin of the source pane.

Note: There are other ways to remove breakpoints while debugging, and we will discuss these as we walk through an example.

Figure 156 shows a breakpoint that is set in the source pane.



```
protected void service(HttpServletRequest req, HttpServletResponse res) throws Se
     res.setContentType("text/html");
     PrintWriter out = res.getWriter();
     out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
     out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>");
     out.println("<H4>This is about as simple a servlet as it gets!</H4>");
     out.println("</BODY><HTML>");
     out.close();
}
```

itso.servjsp.servletapi.SimpleHttpServlet.service(HttpServletRequest, HttpServletResp (2/18/00 5:41:10 PM) Administrator

Figure 156. Breakpoint set in the source pane

Using the Debugger window

We indicated that the Debugger window can be opened by choosing *Window -> Debug -> Debugger*. The Debugger window is also opened if you execute any code that has a breakpoint. You can interact with your program by controlling the program execution flow and by interacting with the threads.

Controlling program execution flow

- ❑*Step in:* Steps into the current statement or method.
- ❑*Step over:* Runs the current statement, and stops before the next statement.
- ❑*Run to return:* Runs the current method, up to the return statement.
- ❑*Resume:* Runs to the next breakpoint, until you manually suspend the thread, or to the end of the program.
- ❑*Run:* Runs the program code, until the next breakpoint or end of program.

Controlling the execution of the program threads

- ❑*Suspend a thread:* To examine a thread at any point while it is running, you must suspend it manually. Threads halted because of a breakpoint or an uncaught exception are suspended automatically.
- ❑*Resume a thread:* Resumes the suspended thread. The program will continue running until it is suspended again or until it terminates.
- ❑*Terminate a thread:* Terminates a thread, and removes it from the debugger.

Debugging a servlet

Now we will walk through the debugger by debugging and stepping through one of our servlet examples.

SimpleHttpServlet servlet changes

You have to modify the *SimpleHttpServlet* class and add a field. This makes it more interesting for the debugger, and allow us to demonstrate an important concept about threading. Add the variable *calledCount* to the class, and add the code snippet to the service method (Figure 157).

```
class:
   private int calledCount;

service:
   ++calledCount;
   out.println("<H4>This servlet has been called: " + calledCount +
               " times.</H4>");
   out.println("</BODY><HTML>");
```

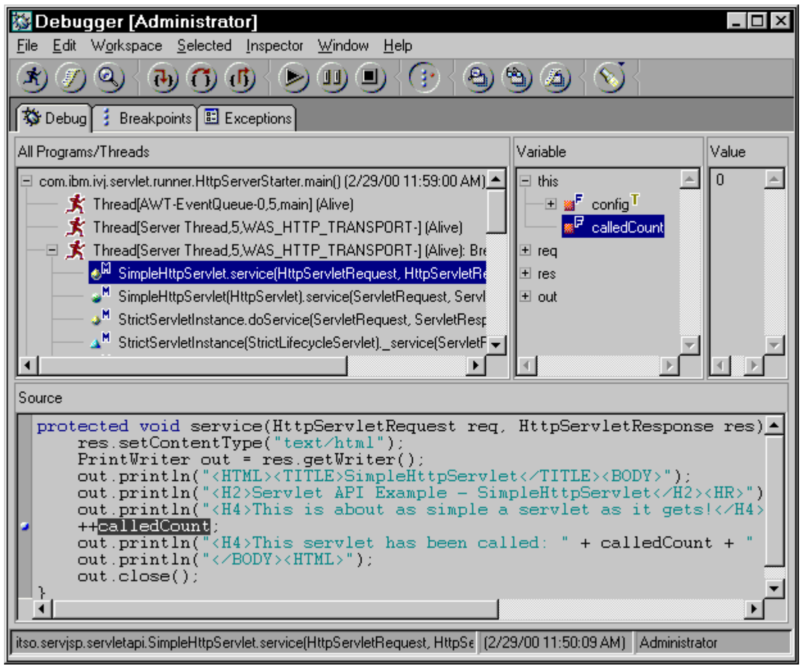
Figure 157. Changes to the simple servlet

Set a breakpoint

Set a breakpoint at the ++calledCount statement.

Run the servlet

Start the *SERunner* class and launch the servlet. The browser window will be launched, but will be waiting for the response from the servlet. You should see the code stop at your breakpoint in the Debugger window (Figure 158).



```
com.ibm.ivj.servlet.runner.HttpServerStarter.main() (2/29/00 11:59:00 AM)
    Thread[AWT-EventQueue-0,5,main] (Alive)
    Thread[Server Thread,5,WAS_HTTP_TRANSPORT-] (Alive)
    Thread[Server Thread,5,WAS_HTTP_TRANSPORT-] (Alive): Bre
        SimpleHttpServlet.service(HttpServletRequest, HttpServletRe
        SimpleHttpServlet(HttpServlet).service(ServletRequest, Servl
        StrictServletInstance.doService(ServletRequest, ServletResp
        StrictServletInstance(StrictLifecycleServlet)._service(ServletF
```

Variable	Value
this	0
config	
calledCount	
req	
res	
out	

```
protected void service(HttpServletRequest req, HttpServletResponse res)
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
    out.println("<H2>Servlet API Example - SimpleHttpServlet</H2><HR>")
    out.println("<H4>This is about as simple a servlet as it gets!</H4>
    ++calledCount;
    out.println("<H4>This servlet has been called: " + calledCount + "
    out.println("</BODY><HTML>");
    out.close();
}
```

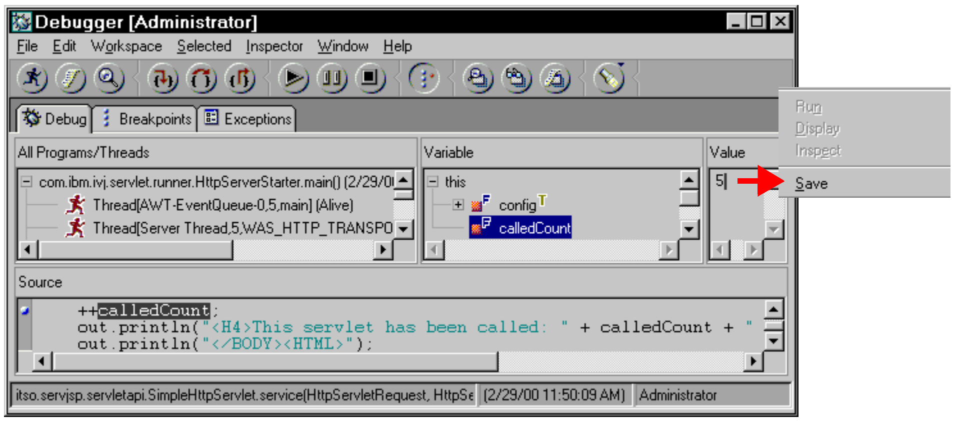
```
itso.servjsp.servletapi.SimpleHttpServlet.service(HttpServletRequest, HttpSe   (2/29/00 11:50:09 AM)   Administrator
```

Figure 158. Debugging the SimpleHttpServlet

Stepping through the program

There are a lot of options you can choose when stepping through the program. You can look at the variable stack, you can suspend the code, you can run till return, or you can step through the program and watch the calledCount get incremented by one.

This program can be called multiple times from the browser just by hitting the refresh button. Each time, it will stop at the selected breakpoint. To see that the debugger is performing incremental compiles, you can update the value of calledCount yourself, and verify that the new value is sent back to the browser. In the debugger window, edit the value of *calledCount*, and select *Save* from the context menu (Figure 159).



```
com.ibm.ivj.servlet.runner.HttpServerStarter.main() (2/29/0
  Thread[AWT-EventQueue-0,5,main] (Alive)
  Thread[Server Thread,5,WAS_HTTP_TRANSPO
```

```
this
  config
  calledCount
```

```
5
```

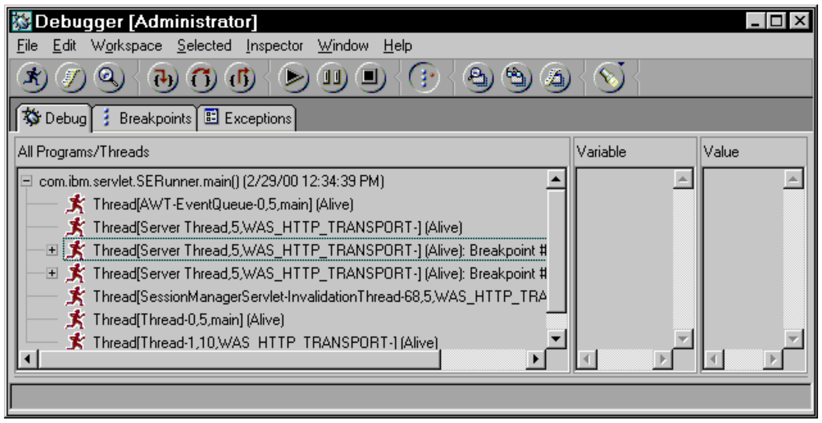
```
++calledCount;
out.println("<H4>This servlet has been called: " + calledCount + "
out.println("</BODY></HTML>");
```

```
itso.servjsp.servletapi.SimpleHttpServlet.service(HttpServletRequest, HttpSe   (2/29/00 11:50:09 AM)   Administrator
```

Figure 159. Changing values while debugging

Working with servlet threads

In the All Program/Threads window pane of the debugger, you can see that there are multiple threads of execution. Many of these threads have to do with the running of the *SERunner* class. In addition, you will see a Thread for each servlet that is running. In the example below, we have triggered *SimpleHttpServlet* from two browser windows. We can see that both threads are running, and have stopped at the breakpoint (Figure 160).


Debugger [Administrator]
File Edit Workspace Selected Inspector Window Help
Debug Breakpoints Exceptions
All Programs/Threads
Variable
Value

```
com.ibm.servlet.SERunner.main() (2/29/00 12:34:39 PM)
  Thread[AWT-EventQueue-0,5,main] (Alive)
  Thread[Server Thread,5,WAS_HTTP_TRANSPORT-] (Alive)
+ Thread[Server Thread,5,WAS_HTTP_TRANSPORT-] (Alive): Breakpoint #
+ Thread[Server Thread,5,WAS_HTTP_TRANSPORT-] (Alive): Breakpoint #
  Thread[SessionManagerServlet-InvalidationThread-68,5,WAS_HTTP_TRA
  Thread[Thread-0,5,main] (Alive)
  Thread[Thread-1,10,WAS_HTTP_TRANSPORT-] (Alive)
```

Figure 160. SERunner Threads

This is a useful technique to show thread interaction among servlets. For instance, we can step through one thread of the *SimpleHttpServlet* and see the calledCount value being incremented, then step on over to the second

SimpleHttpServlet thread, and verify that it has the value set in the first thread.

JSP Execution Monitor

The JSP Execution Monitor enables you to monitor the execution of JSP source, the JSP-generated Java source, and the HTML output. With the JSP Execution Monitor, you can efficiently monitor JSP run-time errors. The JSP Execution Monitor displays the mapping between the JSP and its associated Java source code, and enables you to insert breakpoints in the JSP source.

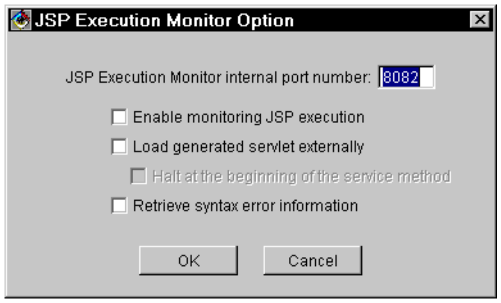
If you find an error in a JSP page, you can also modify the JSP source in a text editor, and then run the JSP source in the JSP Execution Monitor. To load the updated version of the JSP source into the JSP Execution Monitor, you simply have to refresh from the Web browser.

The JSP Execution Monitor highlights the location of syntax errors in both the JSP and JSP-generated Java source.

Launching the JSP Execution Monitor

To launch the JSP Execution Monitor, perform these steps:

❑From the Workspace menu, select *Tools -> JSP Execution Monitor*. The JSP Execution Monitor Option dialog box opens (Figure 161). (The default internal port number for the use of the JSP Execution Monitor is 8082. If port number 8082 is already in use, change the port number in the JSP Execution Monitor internal port number field.)


JSP Execution Monitor Option

JSP Execution Monitor internal port number: 8082

☐ Enable monitoring JSP execution
☐ Load generated servlet externally
☐ Halt at the beginning of the service method
☐ Retrieve syntax error information

Figure 161. JSP Execution Monitor launch window

❑By default, the JSP Execution Monitor mode is disabled. You must select *Enable monitoring JSP Execution* to activate monitoring when a JSP file gets loaded.

❑By default, the *Load generated servlet externally* option is disabled. Selecting this option enables you to load a generated servlet, so that the servlet does not get imported into the IDE. We usually recommend leaving this unchecked because you do not get the class path options that were configured in the WebSphere Test Environment, and your JSPs might not load properly.

Stepping through the JSP

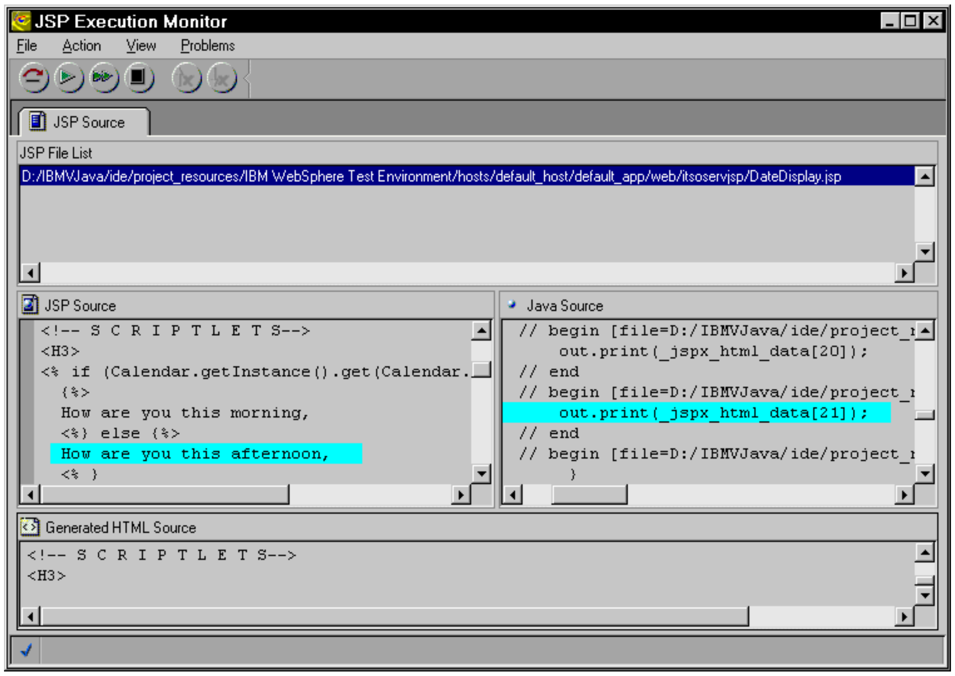
VisualAge for Java ships with a couple of sample JSPs that we can use to test out our configuration, and see that JSPs have been enabled. Follow these steps to run the *DateDisplay* JSP example, and test its result:

❑Start the *SERunne*r process.

❑Enable the JSP Execution Monitor.

❑Enter the following browser command:
`http://127.0.0.1:8080/itsoservjsp/DateDisplay.jsp`

The JSP Execution Monitor window appears and displays the current status of the JSP (Figure 162).



D:/IBMVJava/ide/project_resources/IBM WebSphere Test Environment/hosts/default_host/default_app/web/itsoservjsp/DateDisplay.jsp

```
<!-- S C R I P T L E T S-->
<H3>
<% if (Calendar.getInstance().get(Calendar.
  {%>
  How are you this morning,
  <%} else {%>
  How are you this afternoon,
  <% }
```

```
// begin [file=D:/IBMVJava/ide/project_r
    out.print(_jspx_html_data[20]);
// end
// begin [file=D:/IBMVJava/ide/project_r
    out.print(_jspx_html_data[21]);
// end
// begin [file=D:/IBMVJava/ide/project_r
    }
```

```
<!-- S C R I P T L E T S-->
<H3>
```

Figure 162. JSP Execution Monitor window

Similar to the debugger window for our servlets, you can step through this code, or run to completion. We can also fast-forward and terminate. Using the JSP source and Java source panes, you can see the JSP that was invoked, and the corresponding Java source file that was compiled, and walk through them simultaneously. The HTML output pane shows the JSP response that is generated.

Debugging JSP generated source code

We mentioned earlier that the compiled .java files for JSPs are stored in the file system (WebSphere Test Environment\temp). These files are also imported into the workspace, in the project *JSP Page Compile Generated Code*, and a package named after the \web subdirectory.

JSP compilation occurs when a JSP is invoked the first time (each time after starting the WebSphere Test Environment), or when the underlying JSP file is changed.

Because these servlets exist in the workspace, they are candidates for interactive debugging. You can set breakpoints in the JSP generated source servlets, and debug these servlets in the same way as you debugged the servlet.

You can also step through the code using the JSP Execution Monitor, but this does not give you the ability to interactively change the variable values, or inspect the call stack or threads.

WebSphere Test Environment — advanced configuration

In "Servlet interaction techniques" on page 73 we discussed how servlets can be grouped under a single Web application context. In the VisualAge for Java *SERunner* environment, all servlets and JSPs, by default, belong to the same default Web application. Thus, they share a common *ServletContext*, and can share resources even if we have defined them in different VisualAge projects.

In this section, we describe the WTE *SERunner* default configuration, and how (and where) to locate, build, and/or change servlet resources that your Web application might need. Being part of the same Web application, all servlets and JSPs that are launched through *SERunner* share this same configuration. Keep this in mind when configuring the test environment.

Later, in "WebSphere Test Environment — multiple Web applications" on page 215, we describe how to set up additional Web application environments in VisualAge for Java, by bypassing *SERunner* and running the *ServletEngine* process directly.

Types of resources

Servlets may require additional resources as part of a Web application. These could include active server resources, such as other servlets and JSPs, or passive resources, such as HTML files. Additionally, servlets may require access to servlet configuration files, or other system resources, such as JDBC databases.

Additional servlet examples

Many of the servlet and JSP examples that we have discussed in previous chapters require additional resources. We will describe the basics here so that you can find your way around the WebSphere Test Environment. In the next section, we describe how to configure for specific situations.

Resource locations

In this section, we use `<IBMVJava>` to describe the root path where VisualAge for Java is installed on your system, and `<IBMVJavaWTE>` for the resource directory of the IBM WebSphere Test Environment, for example:

```
<IBMVJava>:     d:\IBMVJava
<IBMVJavaWTE>:  d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment
```

WebSphere Test Environment root locations

These describe the default root locations for the server process:

- *Server root:* `<IBMVJava>\ide\project_resources\IBM WebSphere Test Environment`: This is the server root from which all paths are derived.
- *Default file root:* `<IBMVJava>\ide\project_resources\IBM Servlet IDE Utility class libraries\filename`. When running servlets that perform I/O and you do not specify a path, files will be created here, for example, SaveStats.ser, for a serialized file.

WebSphere Test Environment default application locations

By default, the WTE uses the following directory locations:

- *Document root:* `<IBMVJavaWTE>\hosts\default_host\default_app\web`, is the root directory for HTML and JSP files. For example, `index.html` and `very_simple.jsp` are found here, and are invoked through `http://localhost:8080/very_simple.jsp`.

❑*Document root folders:* You can create additional folders under the document root for specific configurations, for example, `itsoservjsp`.

The URL path for a JSP in this folder would be: `http://localhost:8080/itsoservjsp/myjsp.jsp`.

❑*Compiled JSP:* `<IBMVJavaWTE>\temp\Jsp1_0\default_app\filename`. (For JSP 0.91 the directory is `\temp\default_app\pagecompile\filename`.) This is useful if you want to see the compiled JSP's servlet code.

❑*Class path for servlets:* `<IBMVJavaWTE>\hosts\default_host\default_app \servlets`. This is where the default_app.webapp configuration file can be found.

Configuring project resources

If a servlet requires a configuration file (for example, `impleInitServlet.servlet`, which is an XML servlet configuration file), or a property file, you can place this file anywhere in the SERunner class path. However, we suggest using the following conventions in order to keep your various project resources separate:

❑*Build project specific resource directory root:* `<IBMVJava>\ide \project_resources\ITSO Servlet JSP Redbook`, is the ITSO Servlet JSP Redbook project resources root. This assumes that the ITSO Servlet JSP Project has been added to the SERunner class path.

❑*Build package directories:* You have to create fully qualified directories that match the servlet's package name, for example, `<IBMVJava>\ide \project_resources\ITSO Servlet JSP Redbook\itso\servjsp\servletapi` (`...\itso\servjsp\servletapi\SimpleInitServlet.servlet`).

The four key configuration files

The following four files are the primary files used to configure the WebSphere Test Environment. Many parameters are not applicable to the *SERunner* environment, so we will not go into much detail here. These files will be reintroduced later in this chapter when we discuss configuring for the *ServletEngine* test environment. The configuration matches very closely the configuration in the WebSphere Application Server environment.

SERunner.properties

Location: `<IBMVJavaWTE>\SERunner.properties`.

This is a standard Java property file. You probably do not have to change this file, however, the parameters are:

❑*httpPort:* 8080 (default)

- ❑*docRoot:* `<IBMVJavaWTE>\\hosts\\default_host\\default_app\\web`
- ❑*serverRoot:* `<IBMVJavaWTE>`

default.servlet_engine

Location: `<IBMVJavaWTE>\default.servlet_engine`

This is an XML formatted file. It is the main configuration file for the servlet engine. The key parameters are:

- ❑*Virtual host:* `<websphere-servlet-host name="Host for VisualAge for Java WebSphere Unit Test Environment">`. This tag defines the virtual host in the servlet engine. Only one single virtual host is used in SERunner.
- ❑*webgroup tag:* `<websphere-webgroup name="default_app">`. This tag defines the Web application deployment bindings within the servlet engine. Only the default application (`default_app`) is valid when using *SERunner*.
- ❑*Hostname bindings:* `<hostname-binding hostname="localhost" servlethost= "Host for VisualAge for Java WebSphere Unit Test Environment">`. This tag is for binding a DNS name to a virtual host.
- ❑*MIME types:* This tag defines a mime type mapping for the virtual host.

default_app.webapp

Location: `<IBMVJavaWTE>\hosts\default_host\default_app\servlets\default_app.webapp`

This is the configuration file for the default Web application. The *SERunner* environment supports only this default_app. The key parameters are:

- ❑*Error page:* `<error-page>/ErrorReporter/</error-page>`, the URI page that is called in response to an error during the processing of a servlet; it can be a customized servlet, JSP, or HTML file.
- ❑*Servlet properties:* `<servlet> <name>myser</name> <code>MyServlet</code> <init-parameter> <name>key</name> <value>123</value> <servlet-path>/servlet</servlet-path> <autostart>true</autostart> ... </servlet>`, defines a servlet within a Web application. There can be many servlets defined in this file. A user defined servlet in our application does not have to be defined here (by default, it will be invoked by its class name), but we can use this to specify some additional servlet properties, such as the name (alias) that we use in the browser, and configuration parameters.
- ❑*Invoker servlet:* `<servlet> <name>invoker</name> <servlet-path>/servlet </servlet-path> ... </servlet>`, is a special servlet that allows us to load a class by name, such as itso.servjsp.servletapi.SimpleHttpServlet. The

servlet-path value of */servlet* specifies the URL prefix used to invoke servlets in the browser.

- ❑*JSP:* `<servlet> <name>jsp</name> <code>...</code> </servlet>`, is a special servlet that is used to compile JSPs. The init parameter of *jspemEnable* allows us to enable or disable JSP Execution Monitor support. The class specified in the <code> tag specifies the level of JSP support:
  - `com.ibm.ivj.jsp.runtime.JspDebugServlet                    <== JSP 1.0`
  - `com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet <== JSP 0.91`
- ❑*Error Reporter:* `<servlet> <name>ErrorReporter</name> ...< /servlet>`, is a special servlet that handles the error reporting in our application.

session.xml

Location: `<IBMVJavaWTE>\session.xml`

This controls the WebSphere Session Management functions in the servlet engine. All tags within `<session-data>` are valid in WTE. All others should be ignored.

Configuration for servlet chaining, filtering, and SHTML

This section describes how to configure the WebSphere Test Environment to support servlet chaining, filtering, and the processing of SHTML files. These servlet techniques were discussed in Chapter 4, “Servlets” on page 41.

Servlet chaining

To support servlet chaining, we define a sequence of two or more servlets, such that the response of one servlet is chained as the request into another servlet, until the final servlet in the chain is executed, and the accumulated response sent back to the Web browser. In this way, the execution of all the servlets in the chain produce our composite response.

Figure 163 shows how to add support for servlet chaining. You need to define a *ChainerServlet* in your Web application file, default_app.webapp (or itsoservjsp.webapp). For this example, we are chaining two servlets in the itso.servjsp.servletapi package together to create our composite response.

```
<servlet>
      <name>Servlet Chaining Servlet</name>
      <description>Servlet Chaining Servlet</description>
      <code>com.ibm.websphere.servlet.filter.ChainerServlet</code>
      <servlet-path>/chainer</servlet-path>
      <autostart>true</autostart>
      <init-parameter>
         <name>chainer.pathlist</name>
         <value>/servlet/itso.servjsp.servletapi.ChainerFirst
                /servlet/itso.servjsp.servletapi.ChainerSecond</value>
      </init-parameter>
</servlet>
```

Figure 163. Servlet chaining specification in default_app.webapp

Servlet filtering

In servlet filtering, the servlet changes the MIME type of the response from text/html to a user-defined MIME type, and tie this MIME type to a servlet.

We do not know how to configure the VisualAge for Java Test Environment for servlet filtering.

See “Servlet interaction techniques” on page 426 on how to configure WebSphere Application Server for servlet filtering.

Running SHTML

See “Additional servlet examples” on page 424 for instructions on how to set up WebSphere or VisualAge for Java to run the SHTML example. Basically, you have to associate .shtml files with the JSP 0.91 compiler to have the source compiled into a servlet.

Developing and testing additional servlet and JSP configurations

We have really only covered the very basics of servlet and JSP development, testing, and debugging in VisualAge for Java. This section covers some specific servlet and JSP configurations, primarily to support servlet and JSP interactions discussed in Chapter 4, “Servlets” on page 41 and Chapter 5, “JavaServer Pages” on page 95.

Creating additional servlet examples

We will not provide the detail to create the rest of the servlet examples by hand. You can choose to go through the class creation process for each servlet you create, or you can use code reorganization to copy the first servlet, *SimpleHttpServlet*, using it as a template for your other servlets. You can also import the code from the 5755samp.zip file that is available on the Internet (see Appendix C, “Using the additional material” on page 417.).

To copy a class for a new servlet

❑Select the class file.

❑Select *Reorganize -> Copy* from the context menu.

❑Keep the same package, and select *Rename the copy*. Click on *OK*.

❑Enter the new name, then click on *OK* to generate the new class.

Configuring and running additional servlets

In “Testing the servlets and JSPs” on page 423 (in Appendix C, “Using the additional material”), we provide the detailed steps to allow you to configure and test most of the servlet examples discussed in Chapter 4, “Servlets” on page 41. Some of these servlets have dependencies, such as they may require for supporting HTML pages, initialization files, or servlet configuration files.

The steps to configure and run these servlets can be classified as:

❑Servlet configuration files — servlet initialization parameters and names of called JSPs

❑JDBC database connections—configuring DB2 JDBC connections

❑Redirecting to HTML files — location of HTML files for redirection

❑Redirecting to error_Page — location of error page file

❑Dependencies on generated forms — other servlets

WebSphere Test Environment — multiple Web applications

We mentioned earlier that *SERunner* might not provide all of the functionality required to fully test your application, such as the ability to configure multiple Web applications in the WebSphere Test Environment.

You can run the *ServletEngine* directly, which gives you the following control over your Web application environment:

❑You can run multiple Web applications with their own document root configurations

❑You can set the servlet class path individually for each Web application

❑You can define individual Web application servlet contexts

The following section describes how to configure the environment for a tailored Web application, in addition to the default application.

Configuring multiple Web applications

In this section, we build a new Web application, *itsoservjsp*, to run our examples. The following sections walk us through the steps to configure the *ServletEngine* for two Web applications.

Create new directories

Create the following directories under the WTE root directory `<IBMVJava>\ide\project_resources\IBM WebSphere Test Environment`:

❑`\hosts\default_host\itsoservjsp\servlet`—class path for servlets, and location of the .webapp file

❑`\hosts\default_host\itsoservjsp\web`—document root (for testing, we suggest you include an index.html document in this directory)

❑`\temp\JSP1_0\itsoservjsp`—scratch directory for compiled JSPs (1.0)

You have to create these three directories for each Web application that you define.

Modify default.servlet_engine

Edit the `<IBMVJavaWTE>\default.servlet_engine` file to set up the itsoservjsp Web application. We suggest that you back up this file first prior to making any changes so that you can restore the default *SERunner* environment.

❑Change the WebSphere servlet host name to `default_host`:

```
<websphere-servlet-host name="default_host">
```

❑Add a <websphere-webgroup> for itsoservjsp:

```
<websphere-webgroup name="itsoservjsp">
   <description>ITSO Servlet JSP Redbook</description>
   <document-root>$approot$/web</document-root>
   <classpath>$approot$/servlets$psep$$server_root$/servlets</classpath>
   <root-uri>/itsoservjsp</root-uri>
   <auto-reload enabled="true" polling-interval="3000"/>
   <shared-context>false</shared-context>
</websphere-webgroup>
```

In this definition, we set the `<root-uri>` to `/itsoservjsp`, so that servlets are invoked with `http://localhost:8080/itsoservjsp/servletname`.

❑Change the hostname binding to match the servlet host:

```
<hostname-binding hostname="localhost:8080" servlethost="default_host"/>
<hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
```

With this configuration, we should be able to call HTML files, servlets, and JSP for the itsoservjsp Web application as:

```
http://localhost:8080/itsoservjsp/index.html
http://localhost:8080/itsoservjsp/myPackage.myServletClass
http://localhost:8080/itsoservjsp/myJSP.jsp
```

Create a new itsoservjsp.webapp file

We use the `default_app.webapp` file as our initial template. Copy the file into the new servlets directory (`<IBMVJavaWTE>\hosts\itsoservjsp\servlets`) and rename it as `itsoservjsp.webapp`. This provides us with some basic support. such as the ErrorReporter, Invoker, and JSP servlets.

Customize the `itsoservjsp.webapp` file as follows:

❑Provide a tailored name and a description for the application:

```
<webapp>
   <name>itsoservjsp</name>
   <description>ITSO Servlet JSP Redbook</description>
```

❑Change the JSP servlet to use JSP 1.0 and point to the correct directories for compiled JSPs:

```
<servlet>
   <name>jsp</name><description>JSP support servlet</description>
   <code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code>
   <init-parameter> <name>workingDir</name>
      <value>$server_root$/temp/itsoservjsp</value> </init-parameter>
   ...
   <init-parameter> <name>scratchdir</name>
      <value>$server_root$/temp/JSP1_0/itsoservjsp</value> </init...>
   ...
</servlet>
```

❑Configure one servlet for this application to demonstrate how to specify specific servlet parameters and an alias:

```
<servlet>
   <name>SimpleHttpServlet</name>
   <description>Simple Http Servlet</description>
   <code>itso.servjsp.servletapi.SimpleHttpServlet</code>
   <servlet-path>/simple</servlet-path>
   <init-parameter>
      <name>xxxxxxxx</name> <value>yyyyyyyy</value>
   </init-parameter>
   <autostart>true</autostart>
</servlet>
```

You can later configure additional servlets.

Configuring the ServletEngine class

We will use our existing ITSO Servlet JSP Redbook project for the servlet code. When using multiple Web applications, we have to start the WebSphere Test Environment using the ServletEngine class instead of the SERunner class.

Configuration of ServletEngine

❑In the Workbench, find the IBM WebSphere Test Environment project, select the *com.ibm.servlet.engine* package, and find the *ServletEngine* class.

❑Create a bookmark to this class by selecting the bookmark icon in the top right of your window. This step is optional, but will make it easier to find this class in the future.

Setting command line arguments

Open the *Properties* of the ServletEngine class from the context menu, then select the Program page. Enter following single command line argument:

```
-serverRoot "<IBMVJavaWTE>"
```

For example:

```
-serverRoot "d:\IBMVjava\ide\project_resources\IBM WebSphere Test Environment"
```

Setting the ivj.version

The *SERunner* class sets a system property called *ivj.version* to a non-null value. This tells the WebSphere runtime that it is running in VisualAge for Java and performs a special setup required for this environment. To emulate this behavior when launching *ServletEngine* directly, it is important to set this value to a non-null value. On the same *Properties - Program* page, enter the following line into the *Properties* pane:

```
ivj.version=3.02
```

Setting the class path

In the same Properties dialog, select the *Class Path* page to set up the class path for the ServletEngine.

❑Edit the *Project path* and add the ITSO Servlet JSP Redbook project.

❑Edit the *Extra directories path* and enter all the directories listed in Figure 164. To save you all the typing, you can copy these entries from the same dialog of the SERunner class, and add your Web application servlets directory.

```
../IBM WebSphere Test Environment/lib/db2java.zip;
../IBM WebSphere Test Environment/lib/ns.jar;
../IBM WebSphere Test Environment/lib/ibmwebas.jar;
../IBM WebSphere Test Environment/lib/servlet.jar;
../JFC class libraries/;
../IBM Persistence EJB Library/;
../IBM JSP Examples/;
../Servlet API Classes/;
../IBM Data Access Beans/;
../IBM XML Parser for Java/;
../JSP Page Compile Generated Code/;
../IBM WebSphere Test Environment/;
../IBM IDE Utility local implementation/;
../IBM IDE Utility class libraries/;
D:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\
                            hosts\default_host\itsoservjsp\servlets\;
```

Figure 164. Servlet engine class path directories

Note: This list is dependent on the Version of VisualAge for Java. The best way is to copy the entries from the SERunner class.

Launching *ServletEngine*

To start the *ServletEngine* process directly, select the *ServletEngine* class and *Run -> Run Main*, or click on the Running man icon in the tool bar.

Console window

The VisualAge for Java Console window displays the status of the *ServletEngine* process. If there are problems starting up the environment, messages would be displayed here. The messages that you see for a successful start-up of the *ServletEngine* are shown in Figure 165.

```
Load group: default_app
Instantiate: com.ibm.servlet.engine.webapp.DefaultErrorReporter
009.481 76be ServletInstan A Loading servlet: "ErrorReporter"
009.626 76be WebGroup      A [Servlet LOG]:
"com.ibm.servlet.engine.webapp.DefaultErrorReporter: init"
009.630 76be ServletInstan A Servlet available for service: "ErrorReporter"
......
Instantiate: com.ibm.ivj.jsp.runtime.JspDebugServlet
009.714 76be ServletInstan A Loading servlet: "jsp"
009.784 76be WebGroup      A [Servlet LOG]: "com.ibm.ivj.jsp.runtime.JspDebugServlet:
Scratch dir for the JSP engine is: d:\IBMVJava\ide\project_resources\IBM WebSphere
Test Environment/temp/JSP1_0/default_app
IMPORTANT: Do not modify the generated servlets
009.864 76be ServletInstan A Servlet available for service: "jsp"
Instantiate: com.ibm.servlet.engine.webapp.SimpleFileServlet
009.906 76be ServletInstan A Loading servlet: "file"
009.929 76be WebGroup      A [Servlet LOG]: "com.ibm.servlet.engine.webapp.SimpleFile
Servlet: init"
010.035 76be ServletInstan A Servlet available for service: "file"
Load group: itsoservjsp
Instantiate: SnoopServlet
010.338 76be ServletInstan A Loading servlet: "snoop"
010.421 76be WebGroup      A [Servlet LOG]: "SnoopServlet: init"
010.424 76be ServletInstan A Servlet available for service: "snoop"
.......
Instantiate: com.ibm.servlet.engine.webapp.SimpleFileServlet
010.861 76be ServletInstan A Loading servlet: "file"
010.872 76be WebGroup      A [Servlet LOG]:
"com.ibm.servlet.engine.webapp.SimpleFileServlet: init"
010.874 76be ServletInstan A Servlet available for service: "file"
011.778 76be HttpTransport A HTTP Transport Started on Port 8,080
```

Figure 165. ServletEngine console status

Launching the browser and testing URLs

Start a Web browser, and enter the following URLs to test the Web application configuration:

❑To test that your servlet alias is configured properly, enter `http://localhost:8080/itsoservjsp/simple`

- ❑To test other servlets not defined in the itsoservjsp.webapp file, you must use the fully qualified class name:

  `http://localhost:8080/itsoservjsp/servlet/itso.servjsp.servletapi.SimpleHttpServlet`

Stopping the ServletEngine

To stop the *ServletEngine* process, select the ServletEngine process in the Console window and select *Programs* -> *Terminate* or click on the black square stop button.

Note that killing the ServletEngine is a forceful stop that does not shut down cleanly, and therefore the destroy methods of the servlets are not called. This function will be enhanced in the future.

Using the ServletEngineConfigDumper servlet

The *ServletEngineConfigDumper* servlet is a servlet provided with the WebSphere Application Server. The Java and the class files are in:

```
d:\WebSphere\AppServer\hosts\default_host\examples\servlets
```

You can import this servlet into the VisualAge for Java environment to display additional information about the *ServletEngine* environment.

Import the ServletEngineConfigDumper

You can either copy the class file into the VisualAge for Java directories, or you can import the source code into the Workbench for debugging.

```
Copy: <IBMVJavaWTE>\hosts\default_host\itsoservjsp\ServletEngineConfigDumper.class
```

Configuring the servlet in the Web application

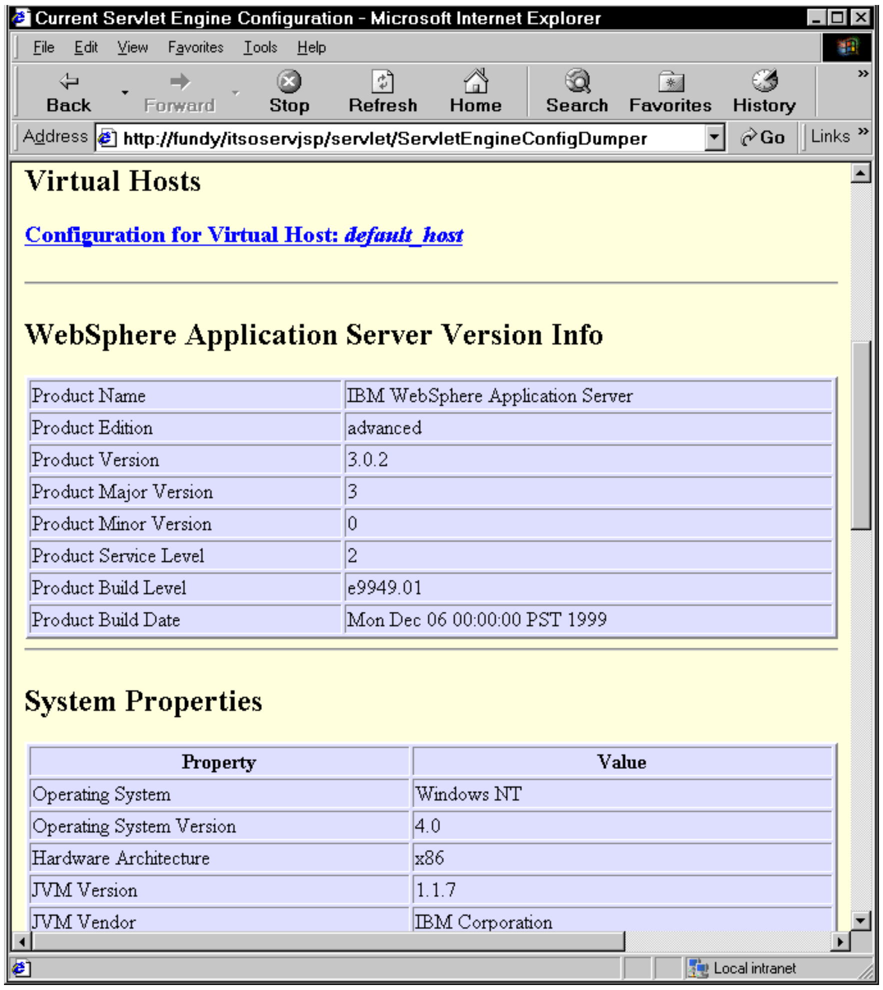
Locate the itsoservjsp.webapp file, and add the following <servlet> configuration:

```
<servlet>
   <name>ServletEngineConfigDumper</name>
   <description>Servlet Engine Configuration Dumper</description>
   <code>ServletEngineConfigDumper</code>
   <servlet-path>/configDump</servlet-path>
   <autostart>true</autostart>
</servlet>
```

Running the servlet

To run this servlet, enter the following URL:
`http://localhost:8080/itsoservjsp/configDump`

Figure 166 shows a partial display of the servlet output that is generated.


Current Servlet Engine Configuration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address http://fundy/itsoservjsp/servlet/ServletEngineConfigDumper
Go Links
Local intranet

Virtual Hosts

Configuration for Virtual Host: *default_host*

WebSphere Application Server Version Info

Product Name	IBM WebSphere Application Server
Product Edition	advanced
Product Version	3.0.2
Product Major Version	3
Product Minor Version	0
Product Service Level	2
Product Build Level	e9949.01
Product Build Date	Mon Dec 06 00:00:00 PST 1999

System Properties

Property	Value
Operating System	Windows NT
Operating System Version	4.0
Hardware Architecture	x86
JVM Version	1.1.7
JVM Vendor	IBM Corporation

Figure 166. ServletEngineConfigDumper output

Restoring SERunner

The *SERunner* process can easily be restored by restoring the original default.servlet_engine file. Any additional Web application directories are ignored.

Configuring and testing servlet and JSP interactions

See Appendix C, “Using the additional material” on page 417 for instructions on how to load the sample code into VisualAge for Java and test the servlets and JSPs.

Support for JavaBeans

VisualAge for Java includes first-class support to create and manage JavaBeans. The class browser has a BeanInfo page where you can define properties, methods, and events for a JavaBean and generate the associated BeanInfo class.

Refer to *“Programming with VisualAge for Java Version 2”*, SG24-5264, for detailed instructions on JavaBeans.

Team development

The software development process is becoming more and more complex. End users are demanding that more function be delivered in less time. Many companies are extending their core business applications to enable new users to work in new ways through their intranet and the Internet, and new applications are required to run on many different platforms. All this often results in the need for large development teams to design, build, and maintain applications. Additionally, the teams are often forced to maintain or expand existing code in a very short time.

Java programmers need development tools that enable them to work together in a highly dynamic environment. They require facilities that easily allow them to manage multiple versions of their work and switch quickly between the different versions. VisualAge for Java Enterprise provides an extremely flexible, productive, and secure built-in team environment for managing the software life cycle process.

The VisualAge for Java team environment is described in detail in the redbook “*VisualAge for Java Enterprise Version 2 Team Support*”, SG24-5245. The team development environment has only minor changes between Version 2 and Version 3; the concepts described in the book are still valid. An extract from the introduction chapter of this redbook is given below.

Overview

At its simplest level, the architecture of the VisualAge for Java Enterprise team environment is a two-tier client/server model: multiple developer workstations connected to a single file server.

Residing on the file server is a shared file, which stores all code for all developers of the development team. This file is called the *repository*.

Each developer workstation has a set of executable files that are common to every client, as well a unique file that contains a single developer's working code set. This file is called the *workspace*.

The client connection to the server is established over a local area network (LAN), and communication between client workspaces and the repository server is through TCP/IP.

Figure 167 shows the VisualAge for Java Enterprise team development environment.

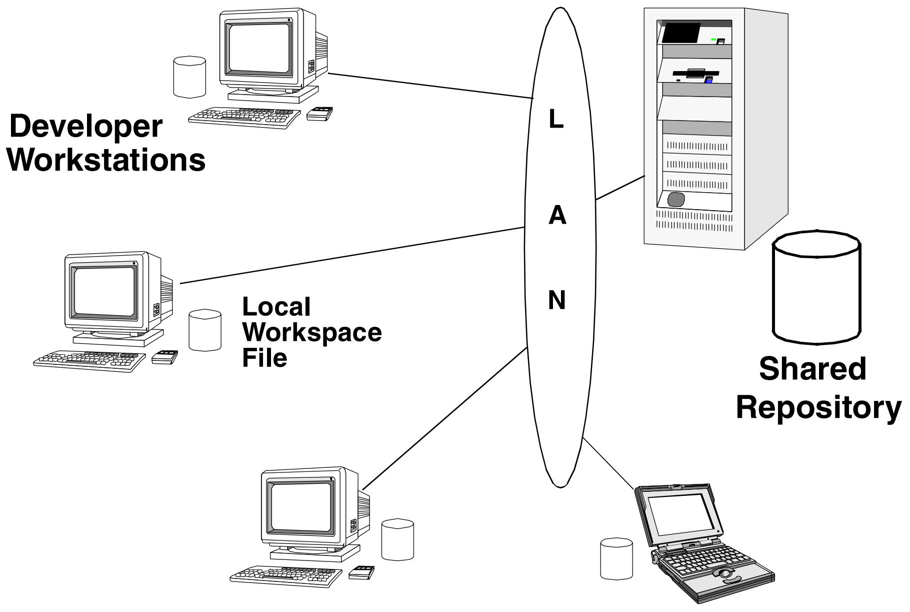


Figure 167. VisualAge for Java Enterprise team development environment

The repository is a large binary file that stores the source and bytecodes of all developer workspaces connected to it. It can be thought of as an object-oriented database that houses all development objects.

The workspace file is unique to each client that is connected to the repository. It contains the bytecodes for the development environment and all program elements that the developer has loaded and is working with. A developer makes changes to code inside the workspace. These changes are always saved immediately into the repository.

Starting VisualAge for Java causes the local workspace file to be loaded into memory and connected to the repository. A VisualAge for Java team client cannot run without a live connection to a repository.

Developers can add program elements, for example, classes or packages, from the repository into their workspace. Only loaded program elements are subject to change by a developer. Generally, many more program elements are stored in the repository than are loaded in a developer's workspace. Developers can also delete program elements from their workspace. Deleted program elements still exist in the repository and can be added back into the workspace.

The workspace file also defines the context of execution when applets and applications are tested during development. All classes and packages that are required to successfully run a program must be loaded into the workspace.

Adding program elements from the repository is a way of easily sharing code among developers working with the same repository. In contrast to other file-based source code management systems, code changes are immediately available to other developers in the group. This does not mean that each developer is directly informed about any changes made by other users. A changed piece of code must be loaded into the workspace in order for it to be accessible. Therefore, each developer has full control over which program elements reside in the workspace.

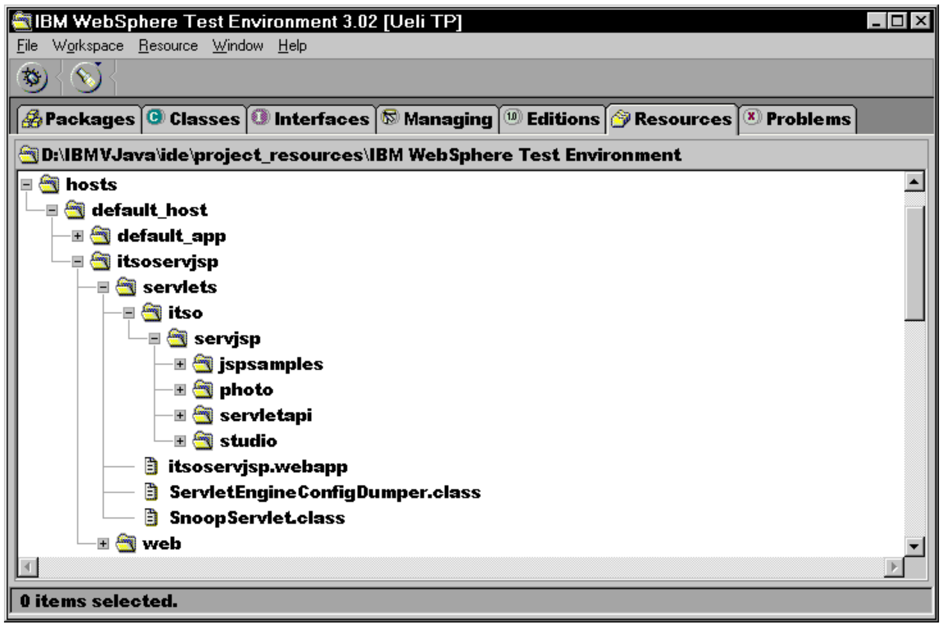
A powerful system of ownership supports the dynamic, concurrent VisualAge for Java team environment. Each program element must have an owner. Thus developers have the freedom and flexibility to make changes and try new things, and at the same time the integrity of each program element is ensured. The ownership model assigns distinct responsibilities to different team members, imposes discipline on the team during the development cycle, and facilitates tracking of changes at maintenance time.

Resource management

Complete applications are composed of Java code and external resources, such as images, sound files, property files, and so forth. The VisualAge for Java repository only handles the Java code. The external resources are stored in the directory structure, and every project has an associated directory, for example:

```
d:\IBMVJava\ide\project_resources\ITSO Servlet JSP Redbook
```

In this directory you would manage the external resource files. If you open a project browser, you can see the list of resources when you select the *Resources* page in the project browser. The *ITSO Servlet JSP Redbook* project does not have any resources, but the *IBM WebSphere Test Environment* project shows many resources (Figure 168).


IBM WebSphere Test Environment 3.02 [Ueli TP]
File Workspace Resource Window Help
Packages Classes Interfaces Managing Editions Resources Problems

```
D:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment
hosts
  default_host
    default_app
    itsoservjsp
      servlets
        itso
          servjsp
            jspsamples
            photo
            servletapi
            studio
        itsoservjsp.webapp
        ServletEngineConfigDumper.class
        SnoopServlet.class
      web
0 items selected.
```

Figure 168. Project resources

You can perform limited function from that window, such as rename, delete, and open. You can associated an external program with specific file types for the *Open* action. Associations are defined in the *Windows -> Options -> Resources -> Resource Associations* dialog.

When you package code into a jar file using the export function, you can select that the resources will be included with the Java and class files.

8 Development with WebSphere Studio

In this chapter, we describe the features and functionality available within WebSphere Studio Version 3.0 and demonstrate how to use this functionality to create, manage, and deploy your Web development projects.

This chapter begins with some background information describing how to use WebSphere Studio to create and manage projects, folders, and files. We then describe, by example, how to use the tools provided in WebSphere Studio to edit project resources, and to add components such as servlets and JavaBeans to your Web pages.

Later in the chapter, we discuss how to deploy your project resources using publishing stages and publishing targets. Then we move on to describe the WebSphere Studio Wizards and how you can use them to easily create Web pages for database and JavaBean interaction.

The final section of this chapter describes how we created a simple application using WebSphere Studio. The example demonstrates how you can easily produce the majority of the code using the wizards, and then customize the generated code to add functionality.

WebSphere Studio overview

The WebSphere Studio application development environment is shown in Figure 169.

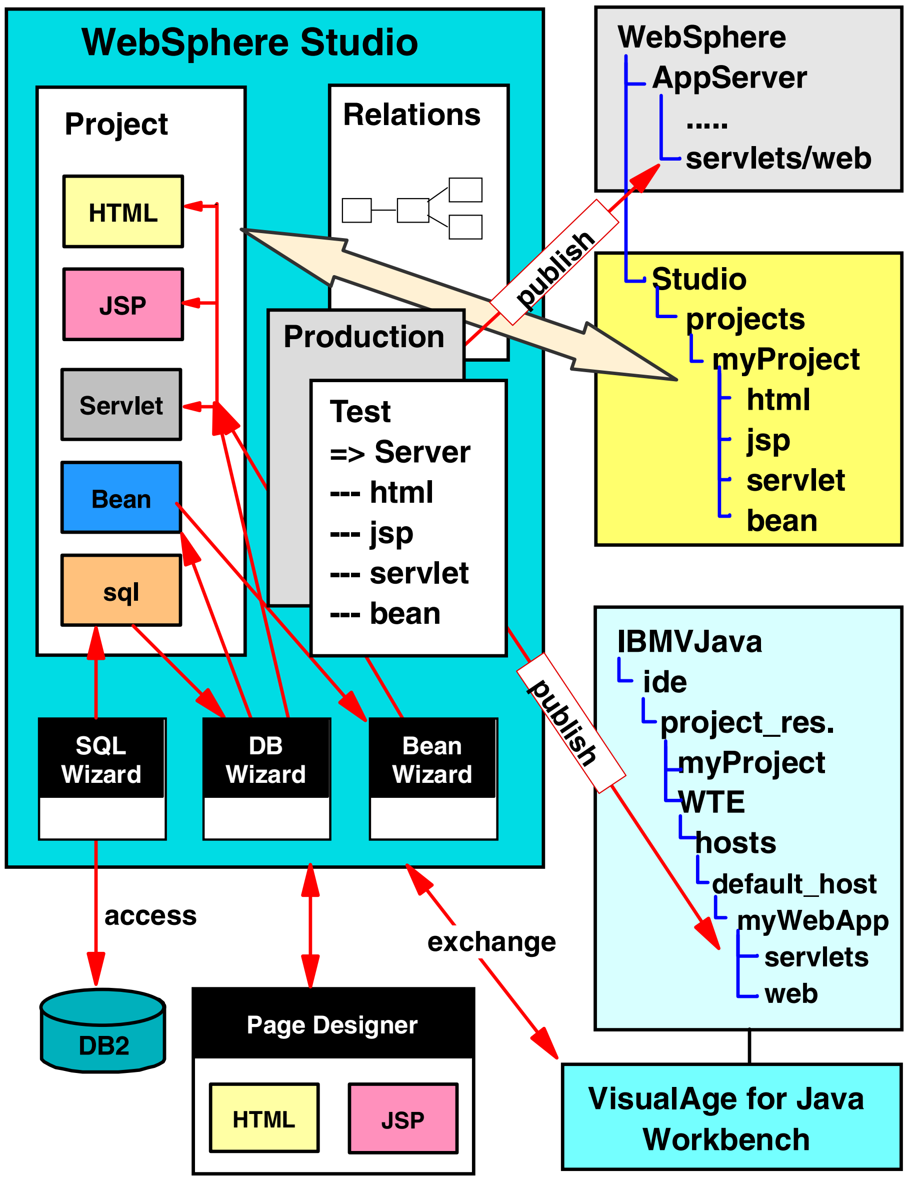


Figure 169. WebSphere Studio application development environment

Here is a short description of the major components of WebSphere Studio:

- ❑WebSphere Studio is a Workbench for developing components of a Web application.
- ❑The user works on a project. Such a project can easily be mapped to a Web application. The Workbench displays the Project view on the left side, and either a Relations view or a Publish view on the right side.
  - The project files are organized into folders at the user's discretion. Normally there is a folder for servlets, and multiple folders for other Web resources, such as HTML, JSPs, and JavaBeans.
  - The Relations view shows how the files are interconnected, for example, an HTML file invokes a servlet, and the servlet invokes a JSP.
  - The Publish views define where files are placed through a publishing action. Normally there are at least two Publish views (Test and Production), but you can define as many as you want. For example, you can map the Test view to place files into the VisualAge for Java WebSphere Test Environment directories, and the Production view into WebSphere Application Server directories.
- ❑WebSphere Studio provides three wizards:
  - The SQL Wizard is used to develop SQL statements based on table definitions in a relational database.
  - The Database Wizard generates HTML, servlet, and JSP code for an SQL statement created with the SQL Wizard.
  - The JavaBean Wizard generates HTML, servlet, and JSP code for a JavaBean.
- ❑WebSphere Studio provides a Page Designer that is used to edit HTML pages and JSPs. The Page Designer generates the HTML and JSP code for static and dynamic Web pages.
- ❑WebSphere Studio can exchange source and class files with the VisualAge for Java Workbench and repository.
- ❑The right side shows the directory structure. WebSphere Studio provides a *projects* directory where individual projects maintain their subdirectory structure.
- ❑A team of developers can share a project directory. When editing files, they are checked out, and only one developer can edit a specific file at a time.

The WebSphere Studio IDE

The WebSphere Studio Integrated Development Environment (IDE) allows you to develop, manage, and deploy Web site resources. The IDE consists of the Project view in the left hand pane and either the Publish view or Relations view in the right hand pane, depending on the selected view option. From any of these panes, you can launch and edit the selected file in its associated editor by *double-clicking* the file icon.

The Publish view shows the folder structure for the selected publishing stage. The Relations view graphically shows the logical links, if any, between the files in your project.

Refer to the WebSphere Studio Guide for further information on the features and functionality of the IDE.

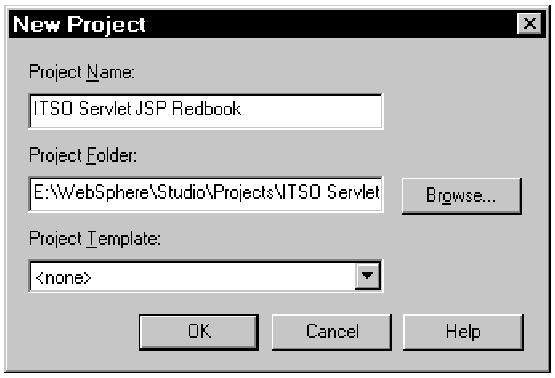
When you start WebSphere Studio the first time, you are prompted to either create a new project or to open an existing project (see Figure 23 on page 35).

Creating a project

WebSphere Studio is a project-based tool which organizes the resources in a project hierarchy. The first task you will want to do in WebSphere Studio is to create a new project that will provide a top-level folder under which all other resources will be placed.

To create a new project, select *File -> New Project*.

The *New Project* window is displayed as in Figure 170. Here you can type a name for the project and select a project template for your project. A directory will be created based on the project name specified in this dialog. WebSphere Studio offers two template project structures, *Corporate1* and *Corporate2*, which include pre-defined folder structures and basic HTML files that provide a good starting point for a typical Web site. You can create your own folder structure by selecting *<none>* for the *Project Template* field.



New Project

Project Name: ITSO Servlet JSP Redbook

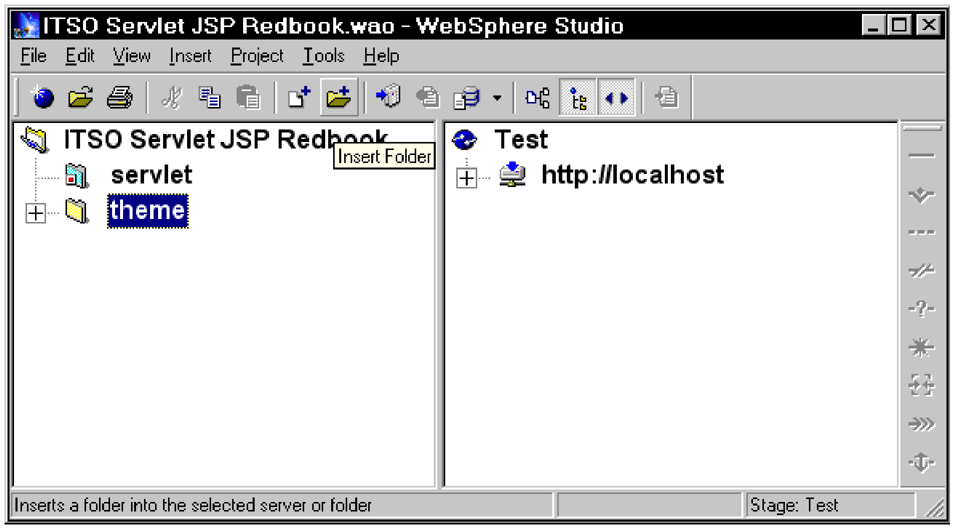
Project Folder: E:\WebSphere\Studio\Projects\ITSO Servlet

Project Template: <none>

Figure 170. Creating a new project

Once the project is created, two folders named *servlet* and *theme* are created by default. Included in the *theme* folder is a cascading style sheet named `Master.css`. Style sheets allow the Web page designer to define the appearance of HTML tags such as <H1> or <BLOCKQUOTE> in one place. Subsequently developed HTML and JSP pages can link to these predefined styles, thus making maintenance of the pages a much easier task. For example, to change the appearance of all <H1> tags in a project, we simply modify the definition within the style sheet.

Figure 171 shows the Studio Workbench for the new project.



ITSO Servlet JSP Redbook.wao - WebSphere Studio

Inserts a folder into the selected server or folder

Stage: Test

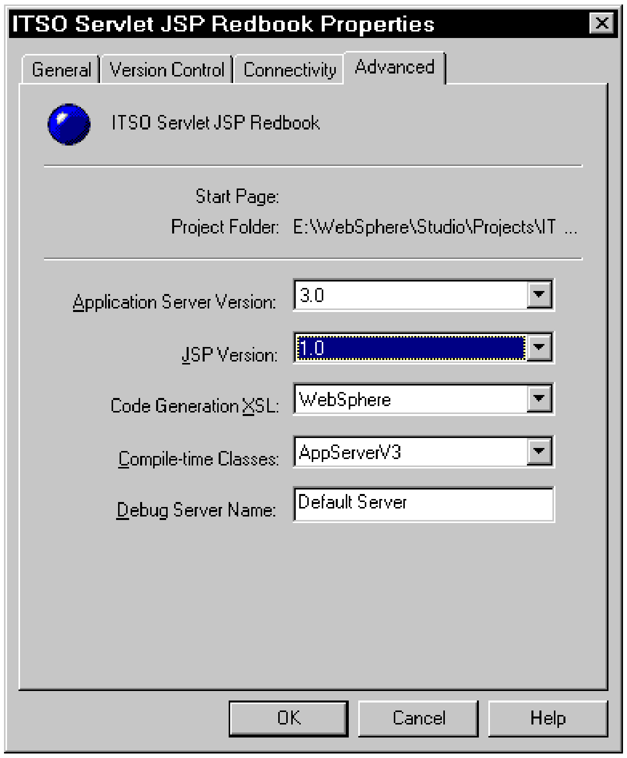
Figure 171. Studio Workbench

Setting the JSP version

WebSphere Studio 3 can be configured to use either JSP 0.91 or JSP 1.0 specifications, depending on which application server you are using. For WebSphere Application Server Version 2, only the JSP 0.91 specification is valid. For WebSphere Application Server Version 3, both JSP 0.91 and JSP 1.0 specifications are valid.

Before starting a new project, you should set the JSP version to the setting appropriate for your project. This is necessary, as the wizards will generate code compatible with the version you specify, and the code is not interchangeable. Also, WebSphere Studio does not allow you to mix JSP 1.0 and JSP 0.91 tags in the same Studio project, or in the same JSP page.

To specify the JSP version, highlight the project node and select *Edit -> Properties* to display the project Properties dialog shown in Figure 172.


ITSO Servlet JSP Redbook Properties

ITSO Servlet JSP Redbook

Start Page:
Project Folder: E:\WebSphere\Studio\Projects\IT ...

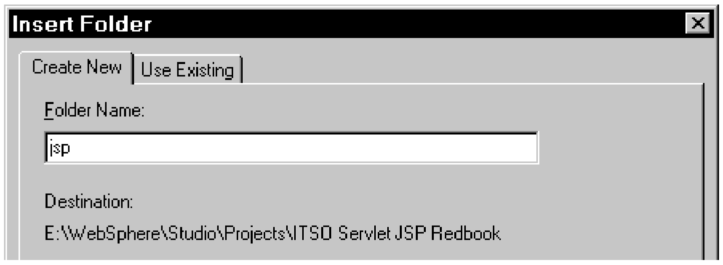
Application Server Version: 3.0
JSP Version: 1.0
Code Generation XSL: WebSphere
Compile-time Classes: AppServerV3
Debug Server Name: Default Server

Figure 172. Set the JSP version in WebSphere Studio

Setting up folders

Defining a folder structure for your project is an important part of organizing project resources into logical groupings. You should consider creating folders which separate HTML files, JSP files, and servlets.

To create a new folder, click the project node and select *Insert -> Folder* (Figure 173). On the *Create New* page, type a name for the folder. This creates a folder under the project level icon. If you want to create nested folders, select a folder, which will be the parent folder, and select *Insert -> Folder.*



Folder Name: jsp

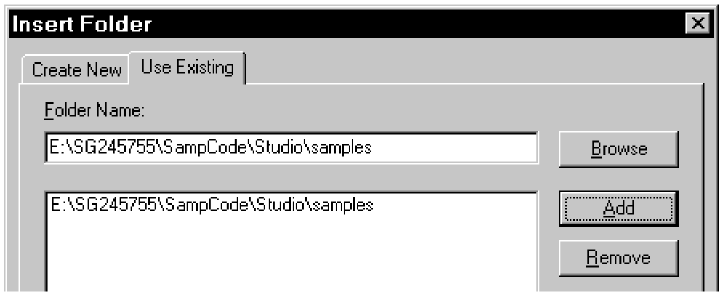
Destination:
E:\WebSphere\Studio\Projects\ITSO Servlet JSP Redbook

Figure 173. Creating a new folder

When you create a new folder in WebSphere Studio, a physical folder is created on the hard disk under the `\projects` directory, for our example:

```
d:\WebSphere\Studio\projects\ITSO Servlet JSP Redbooks\
```

You can also browse for an existing folder on the hard disk for inclusion in the project as shown in Figure 174.



Folder Name: E:\SG245755\SampCode\Studio\samples

E:\SG245755\SampCode\Studio\samples

Figure 174. Inserting an existing folder

Note that when you select an existing folder, the folder and its contents are *copied* into a sub-directory beneath the Studio project directory mentioned above, and the original folder and its contents are not referenced again.

You can insert multiple existing folders by clicking the *Add* button after navigating to each folder.

When the folder structure is complete, you can expand the folders to see the complete structure (Figure 175).

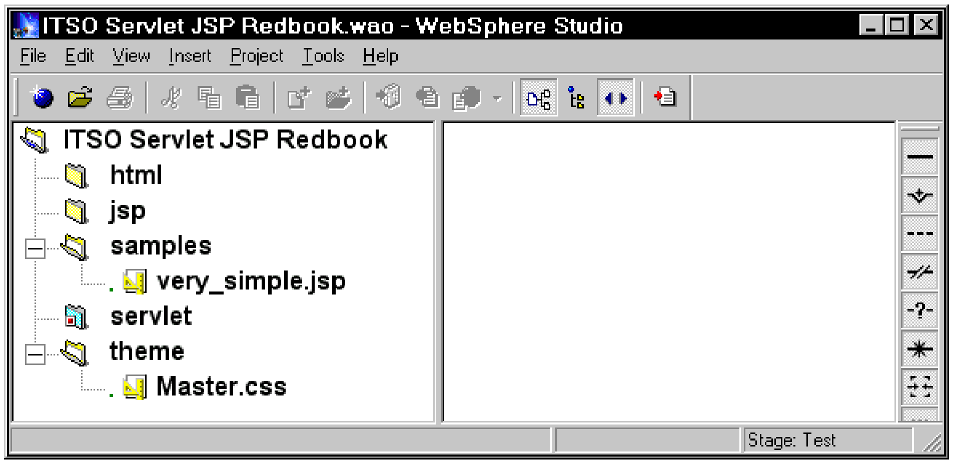

ITSO Servlet JSP Redbook.wao - WebSphere Studio
File Edit View Insert Project Tools Help
ITSO Servlet JSP Redbook
html
jsp
samples
very_simple.jsp
servlet
theme
Master.css
Stage: Test

Figure 175. Completed folder setup

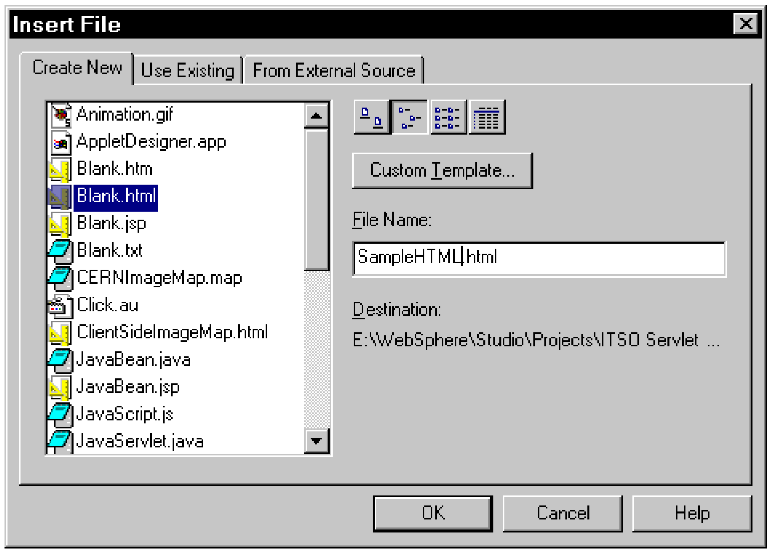
Adding files to the project

The next task you want to do is to create or insert files into the project. Select the folder in the Project file pane where the file will reside, and select *Insert -> File.*

WebSphere Studio provides a number of file templates that you can use as a starting point for your project resources. For example, the HTML file template includes the minimum HTML tags required for a HTML page.

We have added a blank HTML file template named `SampleHTML.html` to the *html* folder, and a blank JSP file template named `SampleJSP.jsp` to the *jsp* folder, using the *Insert File* dialog shown in Figure 176.

The Insert File dialog also enables you to insert existing files (*Use Existing* page) and provides a function to interface to Visual Age for Java (*From External Source* page) to import files directly from the VisualAge IDE (see “Interfacing to VisualAge for Java” on page 291 for details about this feature).



File Name: SampleHTML.html

Destination: E:\WebSphere\Studio\Projects\ITSO Servlet ...

Figure 176. Insert a new file based on a template

At this point, the project now has a JSP file and an HTML file.

We also add the JavaBeans and servlets that were developed in Chapter 5, “JavaServer Pages” on page 95 to enable us to demonstrate how we can use WebSphere Studio to call servlets and embed JavaBeans to create dynamic content.

To add the required Java files and classes, the process is similar to adding an HTML or JSP file except that we select the *Use Existing* tab to navigate to the directories containing the `.class` and `.java` files. You should always place these files underneath the *servlet* folder in the WebSphere Studio project view.

Servlets should also be stored in a folder structure that represents the package name of the class:

```
...\projects\ITSO Servlet JSP Redbook\servlets\itso\servjsp\jspsamples
```

This is a requirement of the JavaBean Wizard. If you do not intend to use the JavaBean Wizard, you do not have to adhere to this structure; however, you will have to set up publishing targets to ensure that these files are placed in the correct folder structure when published to the target server.

At this point, we have a number of files included in our WebSphere Studio project (Figure 177).

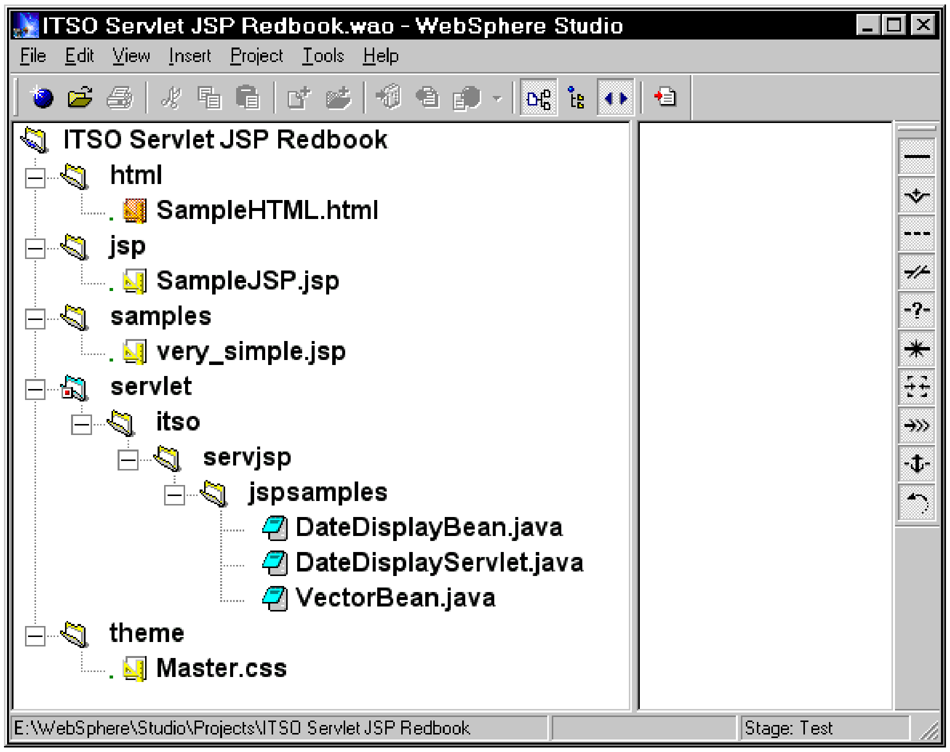


Figure 177. Project structure with folders and files

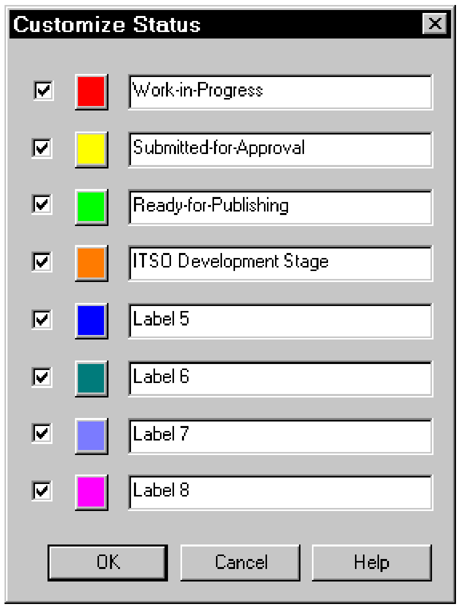
Setting the file status

Before doing anything further with the project, it is often useful to tag the resources in the project with a *status*. Files marked with a status are colored to provide a visual clue which is associated with a particular text label. Typically, you use the status feature to identify a particular phase in the development cycle.

WebSphere Studio offers the following status colors by default:

- ❏Work-in-Progress (Red)
- ❏Submitted-for-Approval (Yellow)
- ❏Ready-for-Publishing (Green)

The status names and colors are fully configurable. You can configure a Status by selecting *Edit -> Set Status -> Customize status* and modifying one of the existing status entries, as shown in Figure 178.


Customize Status

☑	🟥	Work-in-Progress
☑	🟨	Submitted-for-Approval
☑	🟩	Ready-for-Publishing
☑	🟧	ITSO Development Stage
☑	🟦	Label 5
☑	■	Label 6
☑	■	Label 7
☑	🟪	Label 8

Figure 178. Creating a custom status

To apply the new status to selected files in the project pane, select *Edit -> Set Status -> ITSO Development Stage* (or any other predefined status).

Editing project resources

WebSphere Studio allows developers to edit project resources using built-in editing tools. When a resource is being edited, WebSphere Studio flags the resource as *checked-out*, preventing other developers from modifying the same resource in a team environment.

This section describes how to edit resources in WebSphere Studio.

Checking-out and checking-in files

When a resource is edited in WebSphere Studio, the resource is marked as checked-out. When a resource is checked-out, it is locked so that other developers in the team do not have *write* access to it.

Resources that are checked-out have a small red check mark placed in front of the file name, providing a visual clue to other developers as to which resources are checked-out (Figure 177 on page 236 shows the two new files with the mark). This mechanism is very useful in a team environment to prevent simultaneous edits to the same project resource which may result in loss of data. A checked-out file remains that way until a *check-in* option is performed on the file.

Checking out a file is done automatically if the file is edited by launching it from WebSphere Studio. Developers can also manually check-out a selection of files by highlighting them and selecting *Project -> Check Out* (or use the pop-up context menu).

When a resource is checked-out, a copy of the resource is placed in the directory:

```
d:\WebSphere\Studio\check_out\ITSO Servlet JSP Redbook\..folder..\
```

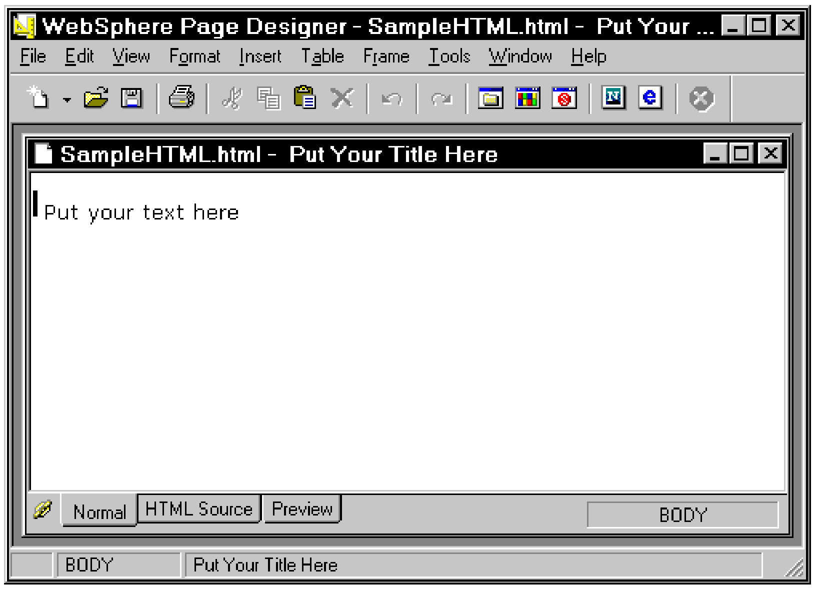
Any editing of the resource is performed on this copy of the file. Only when the file is checked-in does the edited resource get copied back into the original project directory. This feature allows the developer to easily undo any changes made to a file before it is checked-in.

If you want to undo a check-out operation, or you do not want to save any changes made while editing a particular project resource, highlight the files and select *Project -> Undo Check Out*.

When a *check-in* or an *undo check-out* operation is performed, the copy of the resource in the `\check_out` directory is deleted.

Invoking Page Designer

Most graphical project resources can be edited using WebSphere Page Designer. To edit a page resource such as the `SampleHTML.html`, invoke the WebSphere Page Designer by *double-clicking* the file icon in the Project file view WebSphere Studio. This launches the Page Designer and loads the selected file into the Page Designer workspace, as shown in Figure 179.


WebSphere Page Designer - SampleHTML.html - Put Your ...
File Edit View Format Insert Table Frame Tools Window Help
SampleHTML.html - Put Your Title Here
Normal
HTML Source
Preview
BODY
BODY
Put Your Title Here

Put your text here

Figure 179. WebSphere Page Designer

Once the resource is loaded, you can add other HTML page elements using the features provided in WebSphere Page Designer.

Using forms and input fields

Many Web sites contain HTML forms which are used to capture user input. We are going to create a form using the `sample.html` file, which will capture details about the user. Later, we will call a servlet from within this form to process the information inside the form and return the entered information back to the user.

First, we insert a form by selecting *Insert -> Form and Input Fields -> Form*. This creates a form body in which you can insert other input fields. The bounds of the form are denoted by a pink rectangle.

In this example, we create a simple survey form, where we can capture details about the user. You can add a heading to the page by typing inside or outside the form boundary.

We add a large heading titled WebSphere User Survey Form and give it the *Heading 1* format tag by highlighting the text and selecting *Insert -> Paragraph -> Heading1*. We also add a *horizontal rule* tag beneath the heading by selecting *Insert -> Horizontal Rule*.

Next we will use WebSphere Page Designer to insert the user-input fields that will capture user-entered data. This is done by selecting *Input -> Form and Input Fields -> [input field type].* Input fields can be of the types identified in Table 14.

Table 14. Summary of form input fields

Name	Description
Submit button	Used to send form data to the Web server. You can specify a name and caption for the button which can be used to determine which button was pressed in the case of a multi-button form.
Reset button	Used to resent data within the form to default values.
Image button	Used to convert an image to a button and trigger script code for special processing. Does not interact with the server directly.
Push button	Used to trigger script code for special processing. Does not interact with the server directly.
Radio Button	Allows selection of one option from a group of two or more option buttons within the same group.
Check Box	Used to make multiple selections within a particular grouping.
Text Area	Multiple line text entry field.
Text Field	Single-line data entry fields.
List Box	Allow single or multiple selection of values from a list.
Option Menu	Provides a drop-down list of values where only one value can be selected.

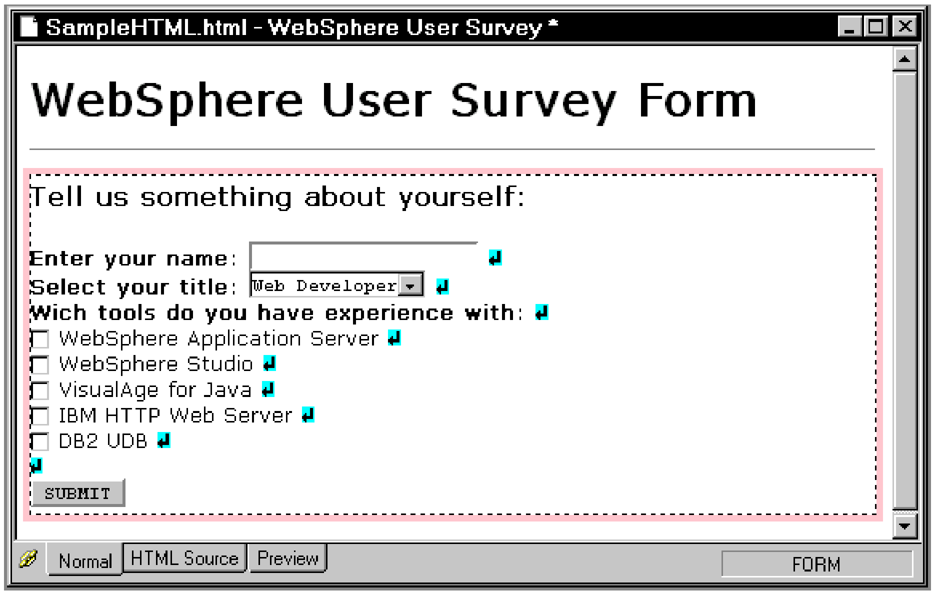
In general, each input field requires that you define a name for the field. For input fields that are grouped, such as radio button and check boxes, you provide one name (a group name) with different values.

For this example, we name the entry field *firstname*, the drop-down list *title*, and all the check boxes *tools*, with values that match the description of the check boxes.

When programming a servlet that will respond to the *Submit* button action, this enables you to iterate through the group and determine which fields are selected within the group. This is described in more detail in “Calling a servlet” on page 241.

The are many attributes and configuration options available for each input type. Refer to the WebSphere Page Designer documentation for detailed information on these attributes.

Building the form is simply a matter of placing the required input fields within the form and providing text labels next to each field in a typical WYSIWYG manner. Figure 180 shows the completed form.


SampleHTML.html - WebSphere User Survey *

WebSphere User Survey Form

Tell us something about yourself:

Enter your name:
Select your title: Web Developer
Wich tools do you have experience with:
WebSphere Application Server
WebSphere Studio
VisualAge for Java
IBM HTTP Web Server
DB2 UDB

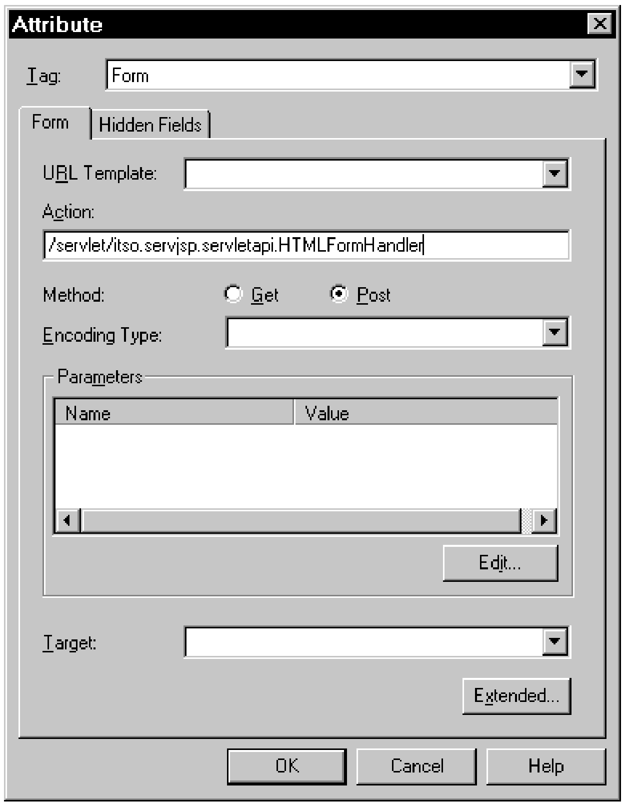
Figure 180. Completed survey form

Calling a servlet

Now that the form is complete, we need a way to send the form data to the Web server for processing. In this example, we call a servlet that iterates through the form elements and performs some actions based on the selections made by the user.

The `itso.servjsp.servletapi.HTMLFormHandler` is provided with the source code examples that accompanies this book.

To modify the form to call the servlet that performs this task, select the form by clicking on its bounding rectangle, and select *Edit -> Attributes* (or double-click). This brings up the attributes dialog for the form, where you can provide a value for the form's action attribute, as shown in Figure 181.



Action: /servlet/itso.servjsp.servletapi.HTMLFormHandler

Method: Get [✓ Post]

Figure 181. Setting the action attribute to call a servlet

Preview the form and view HTML source

You can use the *Preview* tab at the bottom of the survey form in the Page Designer to see how the HTML page might look in a browser.

To view the HTML source code, use the *HTML Source* tab. You can make modifications in the source and they are reflected in the *Normal* view.

Figure 182 shows the source of the HTML page. The actual listing in the Page Designer is color coded, with blue for tags, red for strings, black for text, and purple for keywords.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"><!-- Sample HTML file -->
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>WebSphere User Survey</TITLE>
<LINK href="file:///E:/WebSphere/Studio/Projects/ITSO Servlet JSP Redbook/theme
/Master.css" rel="stylesheet" type="text/css">
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>WebSphere User Survey Form</H1>
<HR>
<FORM action="/servlet/itso.servjsp.servletapi.HTMLFormHandler">
<H3>Tell us something about yourself:</H3>
<P><B>Enter your name</B>: <INPUT size="20" type="text" maxlength="30"
name="firstname"> <BR>
<B>Select your title</B>: <SELECT name="title">
<OPTION selected>Web Architect</OPTION>
<OPTION>GUI Designer</OPTION>
<OPTION>Web Developer</OPTION>
</SELECT> <BR>
<B>Wich tools do you have experience with</B>: <BR>
<INPUT type="checkbox" name="tools" value="WebSphere Application Server"> WebSphere
Application Server <BR>
<INPUT type="checkbox" name="tools" value="WebSphere Studio"> WebSphere Studio <BR>
<INPUT type="checkbox" name="tools" value="VisualAge for Java"> VisualAge for Java
<BR>
<INPUT type="checkbox" name="tools" value="IBM HTTP Web Server"> IBM HTTP Web Server
<BR>
<INPUT type="checkbox" name="tools" value="DB2 UDB"> DB2 UDB <BR>
<BR>
<INPUT type="submit" name="Submit" value="SUBMIT"></P>
</FORM>
</BODY>
</HTML>
```

Figure 182. HTML source view

Save the HTML page and exit the page. Do not close the Page Designer window; it is reused for all editing activities started from the Studio Workbench.

The Web page is now ready to be published to the Web server for testing. As an exercise, use the check-in function on the saved file as a preparation for publishing.

See “Project relationships and integrity” on page 253 for detailed instructions on publishing project resources.

Inserting a JavaBean into a JSP

WebSphere Page Designer provides an interface to easily insert JavaBeans into JSPs. Using the `SampleJSP.jsp` file, we will add the `DateDisplayBean` that is provided with the source code that accompanies this book.

Launch the `SampleJSP.jsp` file from the WebSphere Studio Workbench to load it into the Page Designer. We add labels to the page that identify the properties of the bean that we want to insert, `counter` and `dateString`.

Next we need to declare the JavaBean that we want to use. Before we can insert properties of the bean, it must first be declared. Select a position at the top of the page, before any usage of properties from the bean. Select *Insert -> JSP Tags -> Insert a Bean* (Figure 183).

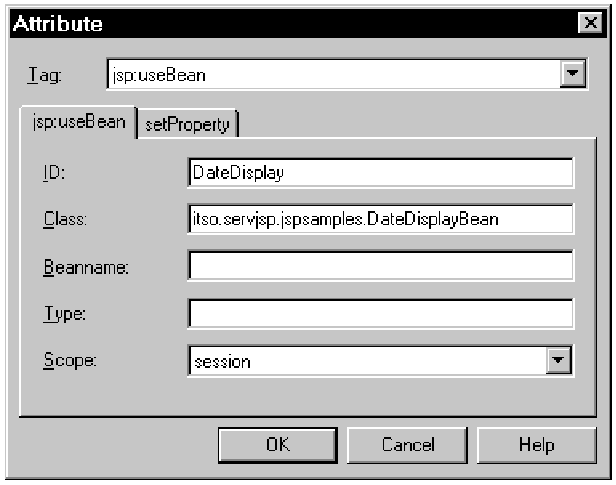
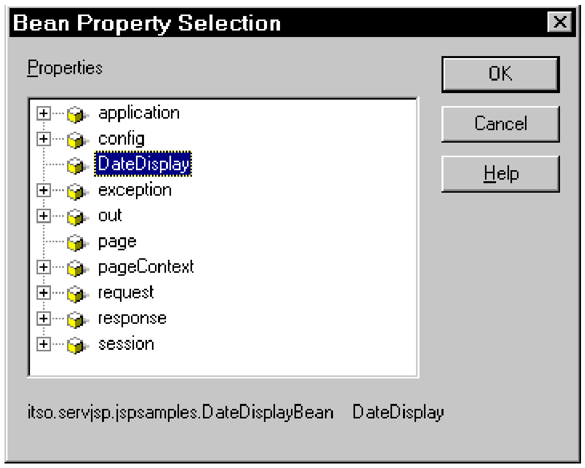


Figure 183. Declaring a JavaBean

In the JSP file, you will notice that a green marker **(J)** identifies the location of the declared bean, and a *jsp:useBean* tag is generated.

Once the bean declaration has been made, you can extract the properties of the bean. Click on where you want the bean property to be inserted and select *Insert -> JSP Tags -> jsp:getProperty Tag* to display the Attribute dialog.

This dialog enables you to specify the bean name and the bean property that you want to insert. Click the *Browse* button to display the list of the available objects and select the `DateDisplay` bean as shown in Figure 184.



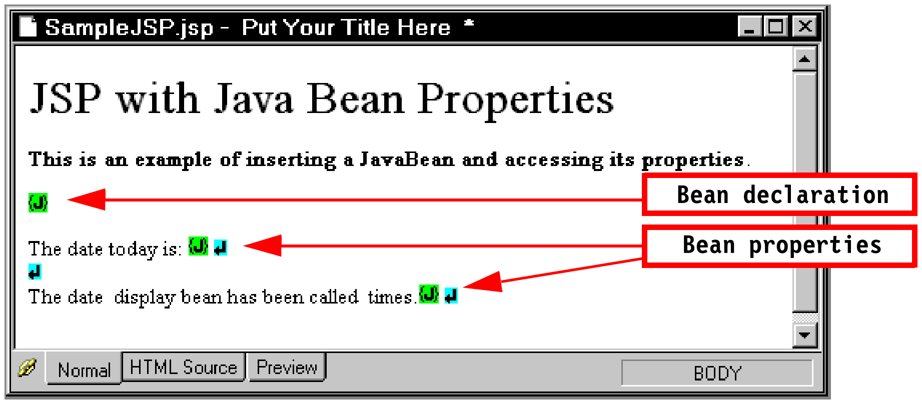
itso.servjsp.jspsamples.DateDisplayBean DateDisplay

Figure 184. Browsing beans and properties

In addition to user-defined objects, this dialog lists other implicit objects available to the page. Refer to Chapter 5, “JavaServer Pages” on page 95 for more information on implicit objects.

Next, in the Attribute dialog, type *dateString* as the name of the property to insert. When complete, notice another green (J) marker embedded in the file where the property is inserted. Repeat this for the *counter* properties that you want to insert into the JSP.

The completed page is shown in Figure 185.



JSP with Java Bean Properties

This is an example of inserting a JavaBean and accessing its properties.

The date today is:

The date display bean has been called times.

Figure 185. Completed JSP including bean properties

By selecting the *HTML Source* tab in WebSphere Page Designer, you can view the source code for the page as shown in Figure 186. This is often useful to see the syntax used to declare and insert bean properties. You can add and edit code directly in the source code view, for example, to change the title.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"><!-- Sample JSP file -->
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css"><TITLE>
JSP with Bean
</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>JSP with Java Bean Properties</H1>
<P><B>This is an example of inserting a JavaBean
and accessing its properties</B>.</P>
<jsp:useBean id="DateDisplay" class="itso.servjsp.jspsamples.DateDisplayBean"
             scope="session" />
<P>The date today is:&nbsp;<jsp:getProperty name="DateDisplay" property="dateString"
/> <BR>
<BR>
The date &nbsp;display bean has been called
&nbsp;times.<jsp:getProperty name="DateDisplay" property="counter" /> <BR>
</P>
</BODY>
</HTML>
```

Figure 186. JSP source

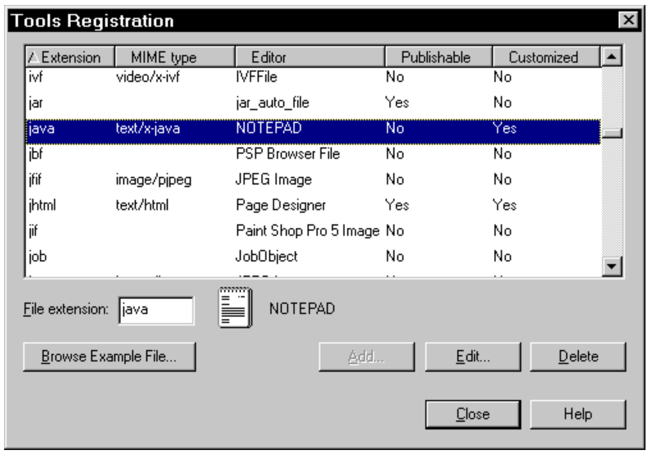
Modifying JavaBeans and servlets

In addition to modifying Web page resources, such as JSP and HTML files, WebSphere Studio allows you to modify Java source code files and recompile the modified source code into class files.

Changing the default editor

The default editor for Java files is `Notepad.exe`. You can change the default editor for a particular MIME time by selecting *Tools -> Tools Registration* (Figure 187).

You can then edit the file with your chosen editor by highlighting the file in the WebSphere Studio Project file view and selecting *Tools -> Edit With -> [your editor]*. You may consider registering VisualAge for Java as an editing tool in addition to your favorite text editor (see “Editing Studio files with VisualAge for Java” on page 390).



Extension	MIME type	Editor	Publishable	Customized
ivf	video/x-ivf	IVFFile	No	No
jar		jar_auto_file	Yes	No
java	text/x-java	NOTEPAD	No	Yes
jbf		PSP Browser File	No	No
jfif	image/pjpeg	JPEG Image	No	No
jhtml	text/html	Page Designer	Yes	Yes
jif		Paint Shop Pro 5 Image	No	No
job		JobObject	No	No

File extension: java NOTEPAD

Figure 187. Tool registration for editing

Compiling source files

If you have made changes to the Java source file and want to recompile it, highlight the source file and select *Compile* (or *Project -> Compile*).

This invokes the Java compiler specified in the *Java* tab in the *Tools -> Preferences* dialog. You can change the Java compiler and class path used by this process by editing the values displayed there.

Publishing stages and publishing targets

WebSphere Studio enables you to set up multiple publishing stages for deployment of WebSphere Studio resources to different locations. You can set up different deployment servers and configurations, depending on how your development environment is structured.

For example, in most development environments, you would probably have, as a minimum, a *test* server and a *production* server. During the development cycle, you would publish your files to the *test* stage, which would reflect a Web server in your test environment. Later, when the project or a project phase is complete, you would publish the files to the *production* stage. Publishing stages are fully configurable, and you can add any number of stages to your project.

WebSphere Studio provides two publishing stages by default, *Test* and *Production*.

- ❑We configure the *Test* stage to publish to the WebSphere Test Environment of VisualAge for Java that runs on the same machine as WebSphere Studio.
- ❑We configure the *Production* stage to publish to the WebSphere Application Server and IBM HTTP Server. We use a real TCP/IP hostname for publishing, although it might actually be the same machine.

Setting up the Test stage

By default, WebSphere Studio provides a configuration for *localhost* that is configured by default to publish all files to the document root of the IBM HTTP Server. Because we want our *Test* stage to publish to the WebSphere Test Environment, we have to modify the configuration of this stage.

To configure the *Test* stage, ensure that your WebSphere Studio project is set to the *Publishing* view by selecting *View* -> *Publishing*. Next, select the test configuration by selecting *Project* -> *Publishing Stage* -> *Test* (Figure 188).

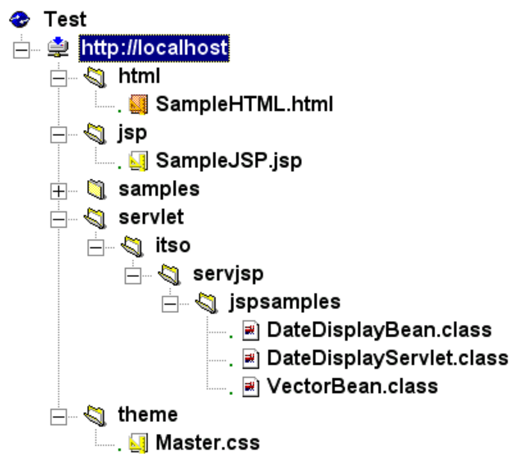
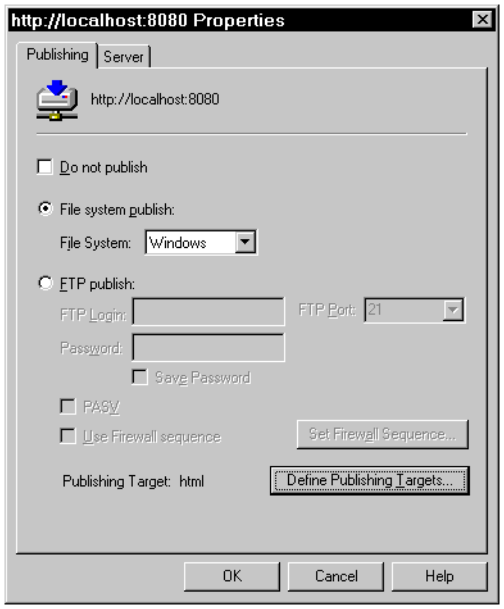


Figure 188. Editing the Test publishing stage

The server name is used for testing directly from WebSphere Studio. The VisualAge WebSphere Test Environment runs (by default) on port 8080, therefore we must redefine the server as *localhost:8080*. Select the Test stage and *Insert* -> *Server*, and enter localhost:8080. Then select each folder under localhost and move (drag) it to localhost:8080. When done, select the localhost server and delete it.

Select the server-level node as shown in Figure 188 and select *Edit -> Properties*. This displays the properties for the *localhost:8080* server (Figure 189).


http://localhost:8080 Properties

Publishing | Server

http://localhost:8080

☐ Do not publish

● File system publish:

File System: Windows

○ FTP publish:

FTP Login: FTP Port: 21

Password:

☐ Save Password

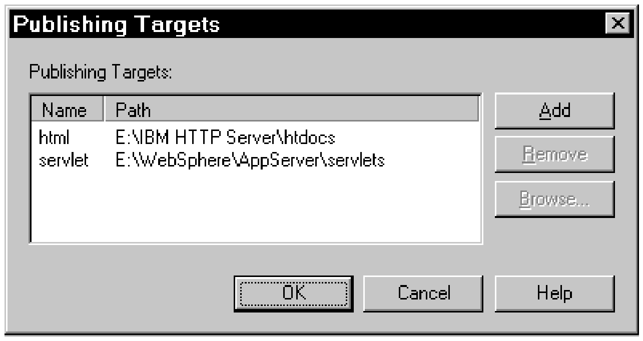
☐ PASV

☐ Use Firewall sequence

Publishing Target: html

Figure 189. Defining publishing properties

Because we will be publishing to the local file system, select the *File system publish* option button and select *Windows NT* as the file system from the drop-down list. Next, we have to configure the publish locations for the files. Select the *Define Publishing Targets* button to display the Publishing Targets dialog (Figure 190).


Publishing Targets

Publishing Targets:

Name	Path
html	E:\IBM HTTP Server\htdocs
servlet	E:\WebSphere\AppServer\servlets

Figure 190. Defining publishing targets for resources

WebSphere provides two publishing targets by default:

- ❑The *servlet* target is used to publish any files in the servlet folder, and points to the WebSphere Application Server if that product is installed on the same machine.
- ❑The *html* target is used for all other project resources in the project, unless you have manually added other publishing targets and linked those to folders in the project. It points to the root document of the HTTP Server if that product is installed on the same machine.

We change the default configuration of the *Test* stage to publish to the WebSphere Test Environment. The publishing paths that we require for our project resources are shown in Table 15.

Table 15. Publishing paths for the WebSphere Test Environment

Resource type (folder name)	**Path**
html	`d:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web`
servlet	`d:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\servlets`

Change the directory for both the *html* and *servlet* entries by clicking on the path and selecting *Browse* to navigate to the appropriate directory.

With this setup, when you publish to the *Test* publishing target, WebSphere Studio creates the folders of your project structure underneath these publishing root directories and copy the files from the project to these folders. You can then start the WebSphere Test Environment from VisualAge for Java and execute the Web pages in the project.

Setting up the Production stage

The process for setting up the publishing targets for the *Production* stage is identical to the *Test* stage. However, there is one step you may have to complete before setting up the publishing targets. WebSphere may not provide a default server (depending on the order of product installation) for the *Production* stage, we have to define the production server.

Display the Production publishing stage by selecting *Project -> Publishing Stage -> Production*. Click the node displaying *Production* and select *Insert -> Server*. Enter the hostname of your WebSphere server:

- ❑We used the hostname *fundy* as our production host
- ❑You can use *localhost* if the server is on the same machine

This defines a new server for which you can now set publishing targets. If your server name is a remote server, you can type a meaningful server name here to identify it.

Provided that you have WebSphere Application Server and Web Server installed locally, WebSphere Studio should default the publishing targets to the correct products, but you may want to change the servlet path to the default application instead of the default servlets directory. Table 16 shows possible options for production publishing targets.

Table 16. Publishing paths for WebSphere application and Web servers

Resource type (folder name)	Path

html	`c:\Program Files\IBM HTTP Server\htdocs`
	`e:\IBM HTTP Server\htdocs`

servlet	`e:\WebSphere\AppServer\servlets` `e:\WebSphere\AppServer\hosts\default_host\default_app\servlets`

On the publishing Properties you can use file system publishing or FTP publishing (Figure 189 on page 249). File publishing is appropriate when the server is accessible through a LAN connection, and FTP publishing is used if the server is remote and not accessible through the file system.

Publishing to a Web application

In a larger environment you will have multiple Web applications defined in WebSphere Application Server, and you want to publish directly to those Web applications. See “Creating your own Web application” on page 135 on how to set up a Web application.

To publish to a Web application, you can modify the standard publishing stages, or you can set up new publishing stages.

Create a new publishing stage

Select *Project* -> *Customize Publishing Stages*, and enter a new name in the *Stage name* field, such as *WebApplication*. Switch the publishing view by selecting *Project* -> *Publishing Stage* -> *WebApplication*.

❑Insert a server as described for the production stage, for example, *fundy*.

❑Define the publishing targets (select *Properties* for the server):

```
html:    E:\WebSphere\AppServer\hosts\default_host\itsoservjsp\web
servlet: E:\WebSphere\AppServer\hosts\default_host\itsoservjsp\servlets
```

The URLs for invoking the resources of a Web application must be prefixed by the name of the Web application. To achieve this, insert a folder in the publishing view under the server (select *Insert -> Folder*), and name the folder *itsoservjsp*. Select all the old folders and drag them into the new itsoservjsp folder. The resulting publishing view is shown in Figure 191.

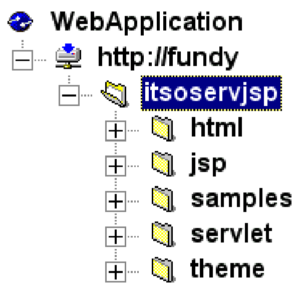
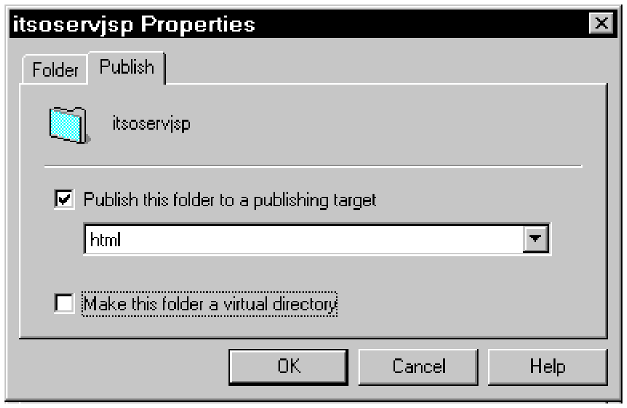


Figure 191. Defining a folder for Web application publishing

Publishing creates subdirectories matching the structure defined in the publishing view, and this creates the wrong structure for Web applications. We have to set up the *itsoservjsp* folder to publish into the proper directory. Select the *itsoservjsp* folder and change its properties (*Edit -> Properties*) as shown in Figure 192.



[✓] Publish this folder to a publishing target

html

[ ] Make this folder a virtual directory

Figure 192. Defining publishing properties for folders

Do not select *Make this folder a virtual directory*, otherwise the itsoservjsp folder name is not inserted into the URL for the browser.

You can also check the servlet folder. It is already set up to publish into the servlet publishing target, overwriting its parent folder.

Manage folder structure

When new directories are added to the project, they may be added to the publishing server, instead of being added to the *itsoservjsp* subfolder.

In such a case, select the new folder in the publishing view and move (drag) it to the itsoservjsp folder.

Publishing to a Web application in VisualAge for Java

Repeat this process and define another publishing stage named *WebApplicationVAJ* for a tailored Web application in the WebSphere Test Environment of VisualAge for Java (see "WebSphere Test Environment — multiple Web applications" on page 215).

Insert a server named *localhost:8080* and define the publishing stages as:

```
d:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\hosts\
            default_host\itsoservjsp\web
            default_host\itsoservjsp\servlets
```

Define the itsoservjsp folder under the server and move all other folders under the itsoservjsp folder. This step is identical to publishing to a WebSphere Web application. You also have to open the *Properties* and select the check box labeled Publish this folder to a publishing target.

Another option is to copy a complete publishing stage to another stage (*Project -> Copy Publishing Stage*) and then just change what is different.

Project relationships and integrity

The parts of a project are interrelated and should be checked periodically for relationship integrity.

Project relationships

For each part in the project you can display its relationship in the Relations view (select *View -> Relations*). For example, when you select the SampleHTML.html file, the relationship diagram shown in Figure 193 is displayed.

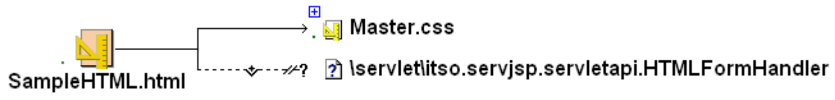
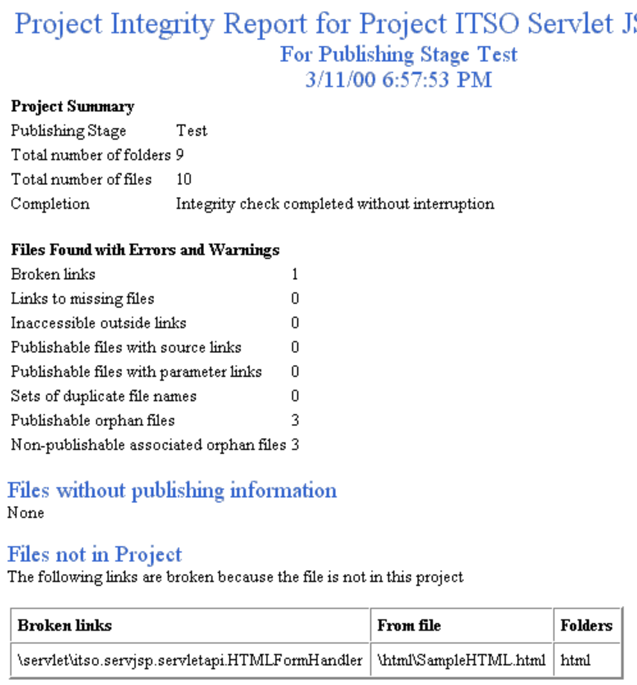


Figure 193. Relationship diagram

This diagram shows one broken link because the SampleHTML.html file invokes the HTMLFormHandler servlet, and we have not added that servlet to the project.

Project integrity

Before publishing files in the project, you should check if there are any broken links within any of the project files. Select *Tools -> Check Project Integrity*. This will generate a report similar to Figure 194, detailing any broken links within the project files. The report is displayed in a Web browser.



Project Integrity Report for Project ITSO Servlet JSP Redbook
For Publishing Stage Test
3/11/00 6:57:53 PM

Project Summary

Publishing Stage	Test
Total number of folders	9
Total number of files	10
Completion	Integrity check completed without interruption

Files Found with Errors and Warnings

Broken links	1
Links to missing files	0
Inaccessible outside links	0
Publishable files with source links	0
Publishable files with parameter links	0
Sets of duplicate file names	0
Publishable orphan files	3
Non-publishable associated orphan files	3

Files without publishing information
None

Files not in Project
The following links are broken because the file is not in this project

Broken links	From file	Folders
\servlet\itso.servjsp.servletapi.HTMLFormHandler	\html\SampleHTML.html	html

Figure 194. Project integrity report

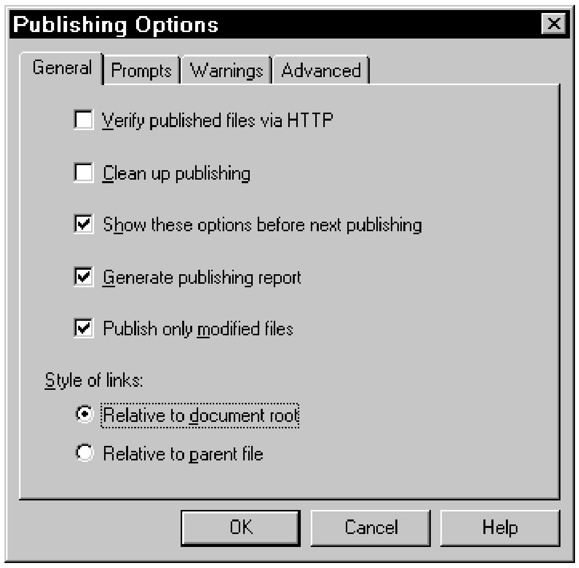
You can fix the broken link by adding the HTMLFormGenerator servlet (Java and class files) to the servlet\itso\servjsp\servlatapi folder.

You may have to invoke the Page Designer and save the HTML file to regenerate the relationship diagram without the broken link.

Publishing a project

Provided that publishing stages and publishing targets have been defined, the project files can be published to any of the publishing stages. WebSphere Studio allows you to publish selected files or the entire project.

To publish selected files to the selected publishing stage, select the file(s) and the select *File -> Publish selected files*. The *Publishing Options* dialog is displayed, allowing configuration of a number of publishing options (Figure 195).



☐ Verify published files via HTTP
☐ Clean up publishing
☑ Show these options before next publishing
☑ Generate publishing report
☑ Publish only modified files

Style of links:
● Relative to document root
○ Relative to parent file

Figure 195. Publishing options

By default, WebSphere Studio only publishes modified files. Most of the configuration options in this dialog relate to the warnings and prompts displayed during the publishing operation. During development, you may find it convenient to turn off all warning and prompts to expedite file publishing.

Clicking the *OK* button starts the publishing operation to the specified publishing stage. Once complete, an HTML formatted report is generated.

Note: We suggest that you select the *Relative to document root* radio button; the other option may create a problem with the style sheet when using the Netscape browser.

Testing published files

To test a published file directly from WebSphere Studio, select *Tools -> Preview File With -> Internet Explorer* (or *Netscape*).

This invokes the selected Web browser with a URL generated for the current publishing stage, for example:

```
http://localhost:8080/html/SampleHTML.html            <= Test
http://fundy/html/SampleHTML.html                     <= Production
http://fundy/itsoservjsp/html/SampleHTML.html         <= WebApplication
http://localhost:8080/itsoservjsp/html/SampleHTML.html <= WebApplicationVAJ
```

This process is very fast and efficient if the Web server is up and running.

WebSphere Studio wizards

WebSphere Studio provides a number of wizards that guide you through the process of creating Java servlets and JavaBeans. The wizards provided in WebSphere Studio are:

- ❑SQL Wizard
- ❑Database Wizard
- ❑JavaBean Wizard

In addition to generating Java source, HTML, and JSP files for you, the WebSphere Studio wizards also invoke the Java compiler where appropriate to produce the class files from the Java source files.

In this section, we will use the DB2 sample database with department and employee information to demonstrate the functionality provided by the wizards.

Code produced by the wizards

When you use WebSphere Studio wizards, by default, the code produced is compatible with WebSphere Application Server Version 2.0. However, if you are using WebSphere Application Server Version 3, you should configure WebSphere Studio to generate code compatible with Version 3 to leverage the enhancements offered in the new version.

WebSphere Application Server Version 3 now supports the JDBC 2.0 Standard Extension API, which provides extensions to support connection pooling. In WebSphere Application Server Version 2, where the JDBC 1.0 specification was implemented, support for connection pooling was only available through special classes provided in the WebSphere environment.

While code generated for Application Server Version 2.0 generally works under Version 3.0, the connection pool support objects used in Version 2.0 are now deprecated and may not be supported in future releases of WebSphere Application Server. For any new code you develop, we recommend that you use the enhancements offered by Version 3.0. You should also consider upgrading your existing code for future compatibility.

To have the wizards generate Version 3.0 compatible code, check that you have selected *3.0* in the *Application Server Version* field in the *Advanced* tab of the project properties dialog (Figure 172 on page 232).

SQL Wizard

The SQL Wizard guides you through the process of building an SQL query that can be used by the Database Wizard to build Web pages. Similar to other visual SQL tools, the SQL Wizard enables you to compose your SQL statements by selecting tables, columns, and operations through a GUI dialog, rather than by typing the statement manually.

In this example, we create a query that lists all employees by department and we will sort the department in ascending order.

Before you access the SQL Wizard, you should first highlight the folder in which the completed `.sql` file will be placed. For example, you can create a folder named sql.

Run the SQL Wizard

To invoke the SQL Wizard, select *Tools* -> *Wizards* -> *SQL Wizard*. The dialog consists of a number of steps.

Welcome page

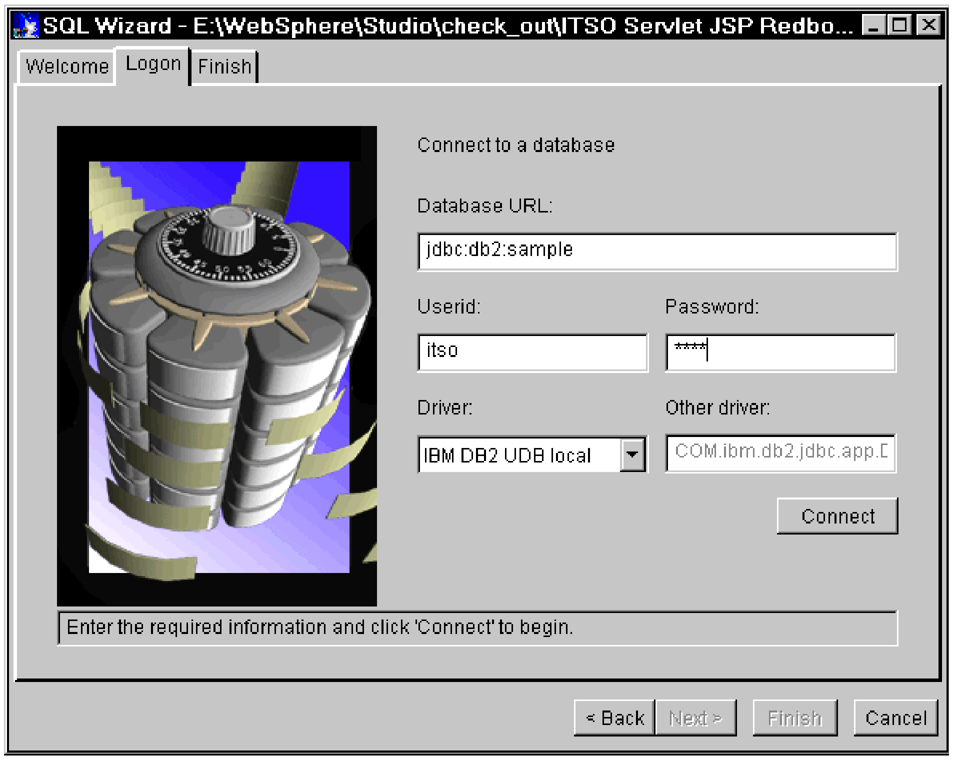
In the Welcome page, specify a meaningful name for the new SQL statement, for example, *AllEmployeesByDept*.

Logon page

Enter the database connection details for the DB2 sample database as shown in Figure 196.

Because the database is installed locally on our machines, we select the *IBM DB2 UDB Local* database driver. Before proceeding, you click the *Connect* button so that the wizard can establish a connection to the database. This allows the wizard to query the structure of the database so that it can be presented to you visually.

You are prompted to select a database schema, and the wizard displays the list of tables. The tables of the sample database are usually under the schema of the user ID that installed DB2. (We used a schema name of USERID for the tables.)


SQL Wizard - E:\WebSphere\Studio\check_out\ITSO Servlet JSP Redbo...

Connect to a database

Database URL: jdbc:db2:sample

Userid: itso

Password: ****

Driver: IBM DB2 UDB local

Other driver: COM.ibm.db2.jdbc.app.D

Enter the required information and click 'Connect' to begin.

Figure 196. SQL Wizard: database logon page

Tables page

Select statement type (Select) and the tables (DEPARTMENT and EMPLOYEE) to be used in the query.

The wizard steps to complete the SQL statement depend on the statement type selected in this page:

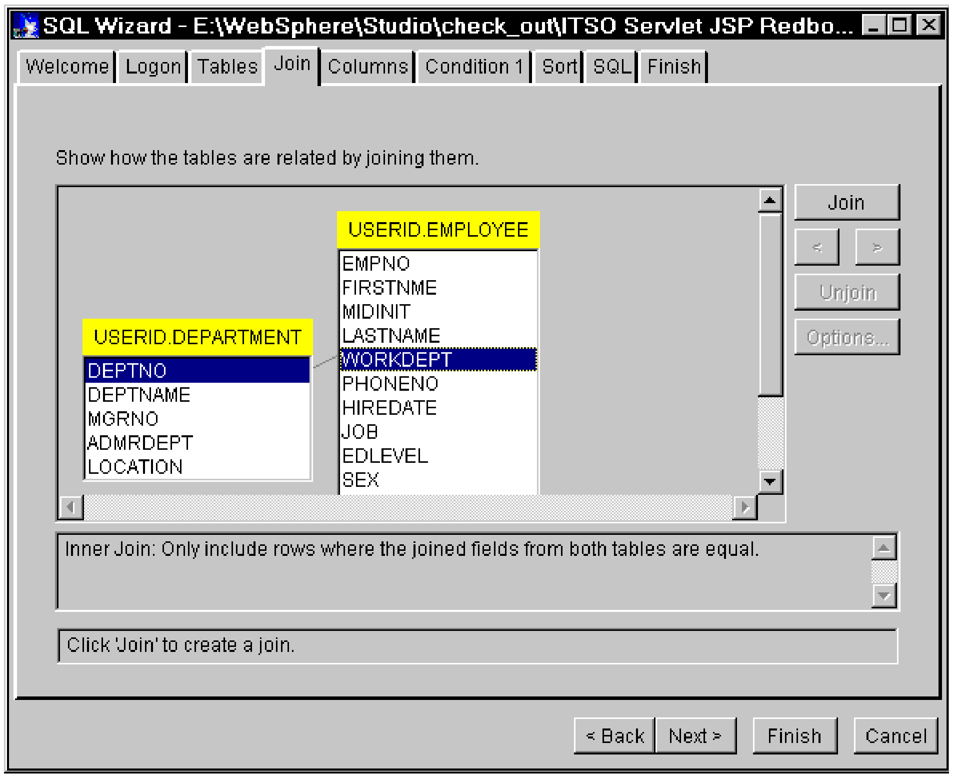
```
Select:    Join, Columns, Conditions, Sort, SQL, Finish
Insert:    Insert, SQL, Finish
Update:    Update, Conditions, SQL, Finish
Delete:    Conditions, SQL, Finish
```

Join page

The Join page allows us to tell the wizard how our tables relate to each other.

From observing the table definitions, we can see that the common field between the two tables is a department number, called DEPTNO in the DEPARTMENT table and WORKDEPT in the EMPLOYEE table.

Join these two tables by selecting the join fields and click on *Join* (Figure 197). A red line is drawn for a successful join operation.



Show how the tables are related by joining them.

USERID.EMPLOYEE
EMPNO
FIRSTNME
MIDINIT
LASTNAME
WORKDEPT
PHONENO
HIREDATE
JOB
EDLEVEL
SEX

USERID.DEPARTMENT
DEPTNO
DEPTNAME
MGRNO
ADMRDEPT
LOCATION

Inner Join: Only include rows where the joined fields from both tables are equal.

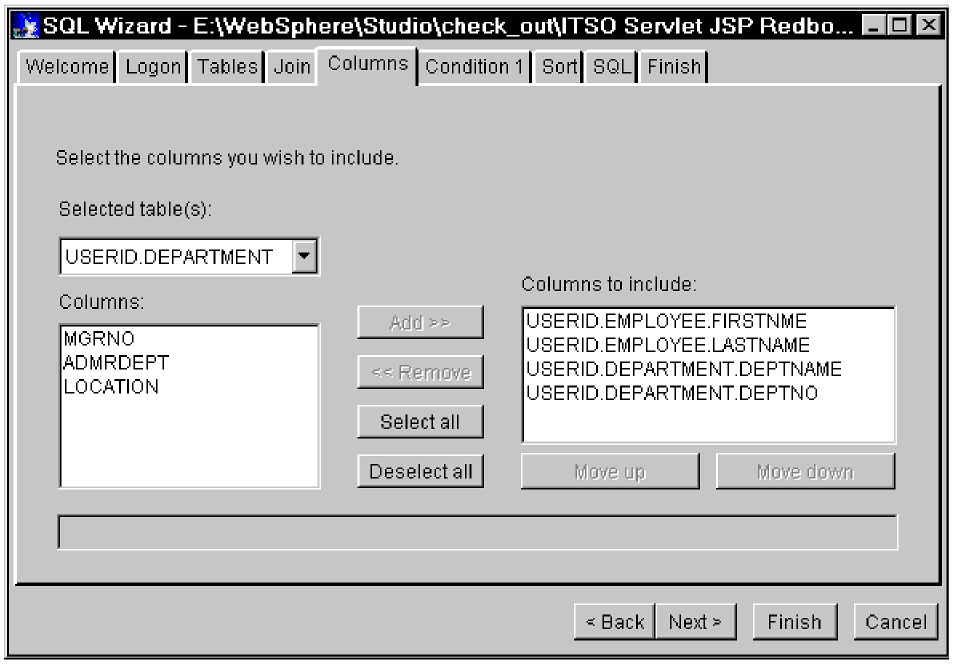
Click 'Join' to create a join.

Figure 197. SQL Wizard: joining tables

You can specify the type of join by clicking the *Options* button to display the Join Properties dialog. In this example, the default join type of *Inner Join* is appropriate for the way we want to present our data, but outer joins are supported as well. Consult the SQL or DB2 documentation for information on other types of joins.

Columns page

On the Columns page you select the columns that you want to include in the result set produced by the query. For this example, select the columns shown in Figure 198. You can change the order in which the column data is placed in the result set by clicking the *Move up* and *Move down* buttons.



Select the columns you wish to include.

Selected table(s): USERID.DEPARTMENT

Columns:
MGRNO
ADMRDEPT
LOCATION

Columns to include:
USERID.EMPLOYEE.FIRSTNME
USERID.EMPLOYEE.LASTNAME
USERID.DEPARTMENT.DEPTNAME
USERID.DEPARTMENT.DEPTNO

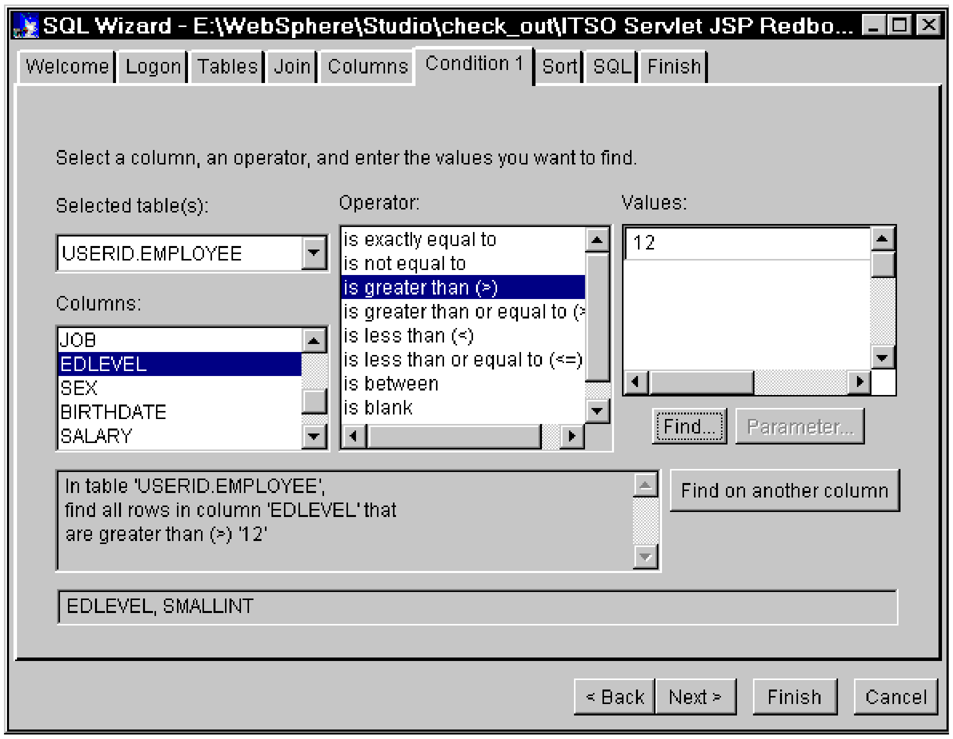
Figure 198. SQL Wizard: selecting columns

Condition page

Conditions enable you to specify restrictions on the data retrieved by the query. You can specify conditions for all statement types other than *Insert*. Adding a condition builds a WHERE clause to the SQL statement using the columns, operators, and values specified in this screen.

For example, we could add a condition to the select query to retrieve only employees over a given education level, as shown in Figure 199. You can use the *Find* button to query the table for values.

Similarly, we could add a condition to an update statement to change only employees below a given salary.



Select a column, an operator, and enter the values you want to find.

Selected table(s): USERID.EMPLOYEE

Columns: JOB, EDLEVEL, SEX, BIRTHDATE, SALARY

Operator: is exactly equal to, is not equal to, is greater than (>), is greater than or equal to (>, is less than (<), is less than or equal to (<=), is between, is blank

Values: 12

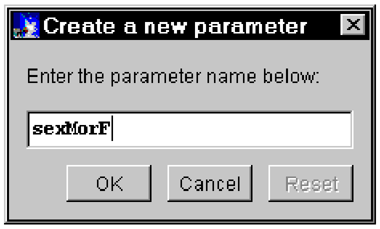
In table 'USERID.EMPLOYEE',
find all rows in column 'EDLEVEL' that
are greater than (>) '12'

EDLEVEL, SMALLINT

Figure 199. SQL Wizard: specifying conditions

You can add values either by hard-coding the value or by specifying a *parameter*. If you specify a parameter, the Database Wizard generates HTML input fields to capture the parameters from the user.

Click on *Find on another column* to add another Condition page. For example, we want to retrieve only male or female employees, by user input. To specify a value by parameter, click the *Parameter* button to display the *Create a new parameter* dialog as in Figure 200.



Enter the parameter name below:

sexMorF

Figure 200. SQL Wizard: condition parameter

When adding parameters, each parameter name must be unique and must not have been used in any prior step in the wizard. There are many operators available in the *Conditions* screen that determine the type of filtering to perform in the query. Depending on the selected operator, the values you have to specify will vary.

Sort page

On the Sort page, you can order the results of the query. This step builds an ORDER BY clause to the SQL statement.

For our example, select *Ascending* sort order and add the DEPTNAME and LASTNAME columns. You must select the *Sort order* before you add a column.

SQL page

The SQL page displays the accumulated SQL query that the wizard has generated (Figure 201).

```
SELECT
      USERID.EMPLOYEE.FIRSTNME,
      USERID.EMPLOYEE.LASTNAME,
      USERID.DEPARTMENT.DEPTNAME,
      USERID.DEPARTMENT.DEPTNO
FROM
   USERID.DEPARTMENT,
   USERID.EMPLOYEE
WHERE
   (
      (
        USERID.DEPARTMENT.DEPTNO = USERID.EMPLOYEE.WORKDEPT    <=== join
      )
     AND
    ( (
         USERID.EMPLOYEE.EDLEVEL > 12                        <=== condition
      )
   AND
      (
         USERID.EMPLOYEE.SEX = ?                             <=== parameter
      ) )
   )
ORDER BY
   USERID.DEPARTMENT.DEPTNAME,
   USERID.EMPLOYEE.LASTNAME
```

Figure 201. SQL Wizard: generated SQL

Testing the SQL statement

From this page you test that the query works by clicking on *Run SQL*. You are prompted for the parameter (enter M or F), and the query is run and the results are displayed.

You can also copy the contents of the query to the clipboard. This is useful if you want to paste the SQL query into the DB2 Command window.

Finish page

Clicking the *Finish* button completes the wizard and creates the `.sql` file in the folder you selected before invoking the wizard. This file includes details of the SQL statement in addition to other information required by WebSphere Studio.

Insert page

The Insert page enables you to specify values to be inserted into a table. This screen is only presented if you have select the *Insert* statement type in the Tables page. Data can be entered as hard-coded values or as parameters.

To create a parameter field, click the *Parameters* button and type a name for the parameter. The names entered will correspond to the names of the Bean properties generated by the Database Wizard. For example, if there is a database column named MGRNO, and a parameter named *ManagerNumber* is created for this field, the bean created by the Database Wizard contains a property called ManagerNumber. Similarly, any HTML pages generated by the Database Wizard will use the specified parameter name as the name for the HTML input field and its associated label.

Update page

The Update page enables you to specify column values and conditions for the Update statement. Similar to the Insert statement type, you can hard-code values or use parameters.

Changing the SQL statement

The query can be modified at any time using the SQL Wizard by double-clicking on the file. Although you can view the .sql file in an editor, you cannot modify the SQL statement produced by the SQL Wizard by manually changing the .sql file.

Database Wizard

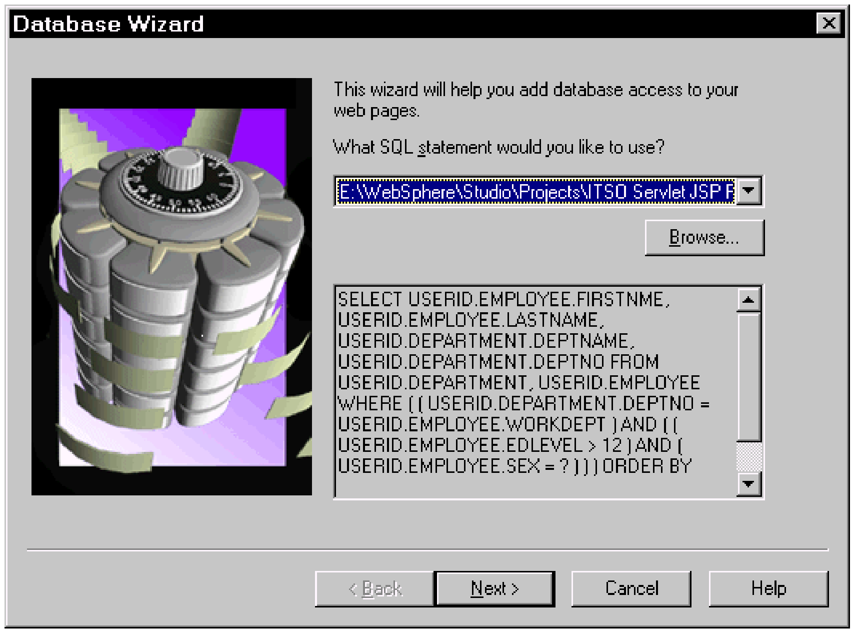
The Database Wizard uses `.sql` files created by the SQL Wizard to generate input pages, results pages, error pages, and the corresponding JavaBeans and servlets responsible for performing the database interaction.

Run the Database Wizard

Start the wizard from the *Tools* menu.

SQL statement selection

In the first step of the Database Wizard, you select the `.sql` file to be used. You can preselect the .sql file before starting the wizard, or you can use the Browse button to find it. This page also displays the SQL statement corresponding to the selected file (Figure 202).



This wizard will help you add database access to your web pages.

What SQL statement would you like to use?

E:\WebSphere\Studio\Projects\ITSO Servlet JSP F

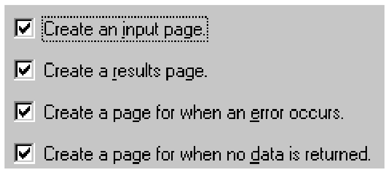
```
SELECT USERID.EMPLOYEE.FIRSTNME,
USERID.EMPLOYEE.LASTNAME,
USERID.DEPARTMENT.DEPTNAME,
USERID.DEPARTMENT.DEPTNO FROM
USERID.DEPARTMENT, USERID.EMPLOYEE
WHERE ( ( USERID.DEPARTMENT.DEPTNO =
USERID.EMPLOYEE.WORKDEPT ) AND ( (
USERID.EMPLOYEE.EDLEVEL > 12 ) AND (
USERID.EMPLOYEE.SEX = ? ) ) ) ORDER BY
```

Figure 202. Database Wizard: SQL statement selection

Note: You get an error message box if your project name contains blanks or invalid characters. The default package name used for the generated code is the project name, and invalid characters are eliminated. You can overwrite the package name before generating the code.

Web Pages

On this page you select the Web pages that the wizard should produce.


☑ Create an input page.
☑ Create a results page.
☑ Create a page for when an error occurs.
☑ Create a page for when no data is returned.

The available options are shown in Table 17.

Table 17. Web pages generated by the Database Wizard

Page Type	**Description**
Input page	A Web page containing an HTML form with user input fields. If your query contains parameters, you want to generate an input page to capture the parameter data from the user. You can also capture database connection information such as user ID and password.
Results page	For a *Select* query, this page displays the data returned by the query. You can specify whether to return the results in table or list form. For *Insert* and *Update* queries, you can choose to display the values used in the Insert or Update operation and optionally display the number of rows affected by the query.
Error page No Data page	These pages are displayed when an unexpected error occurs or when no data is returned from a query. The pages may be existing pages you have created or you can let the wizard create them for you. For *Insert* and U*pdate* queries, the No Data page is unavailable.

Input page

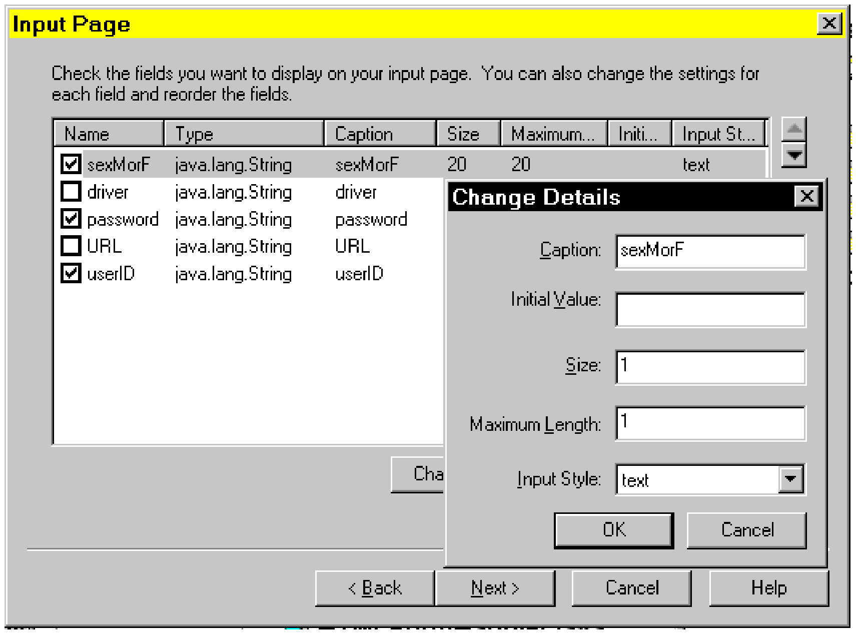
Here you specify the inputs fields to be included in the input HTML page generated by the wizard. The fields that are presented are the parameters of the SQL statement and database connection information (Figure 203).

To customize the behavior of the input fields created by the wizard, select the field that you want to customize and click the *Change* button to display the *Change Details* dialog shown in Figure 203.

You may want to change the length of the fields, for example, the employee sex field is only 1 character, and user ID and password are 8 characters.

Select as many fields as you want included on your data entry page. As a minimum, you would include all fields specified as parameters during in the

SQL Wizard. Remember that other than the database connection fields, only those fields identified as parameters in the SQL Wizard are available for selection.


Input Page

Check the fields you want to display on your input page. You can also change the settings for each field and reorder the fields.

Name	Type	Caption	Size	Maximum...	Initi...	Input St...
☑ sexMorF	java.lang.String	sexMorF	20	20		text
☐ driver	java.lang.String	driver				
☑ password	java.lang.String	password				
☐ URL	java.lang.String	URL				
☑ userID	java.lang.String	userID				

Change Details

Caption: sexMorF

Initial Value:

Size: 1

Maximum Length: 1

Input Style: text

Figure 203. Database Wizard: input fields

Results page

You can configure the way the results are displayed to the user in the Results page. The page produced by the Database Wizard is displayed to the user after execution of the SQL query or statement.

For all queries, you can chose to include any of the parameters specified during the SQL Wizard. In addition, you may chose to display information fields, such as the SQL statement text or the connection information.

Results from a *Select* query can be presented either in table format or as a drop-down list:

❑If you select the *List* format, a drop-down list is created for each column specified in the *Select* query. For example, Figure 204 shows the results of a *Select* statement where department name and employee last name are displayed.

❑If you select the *Table* format, results are displayed in an HTML table.

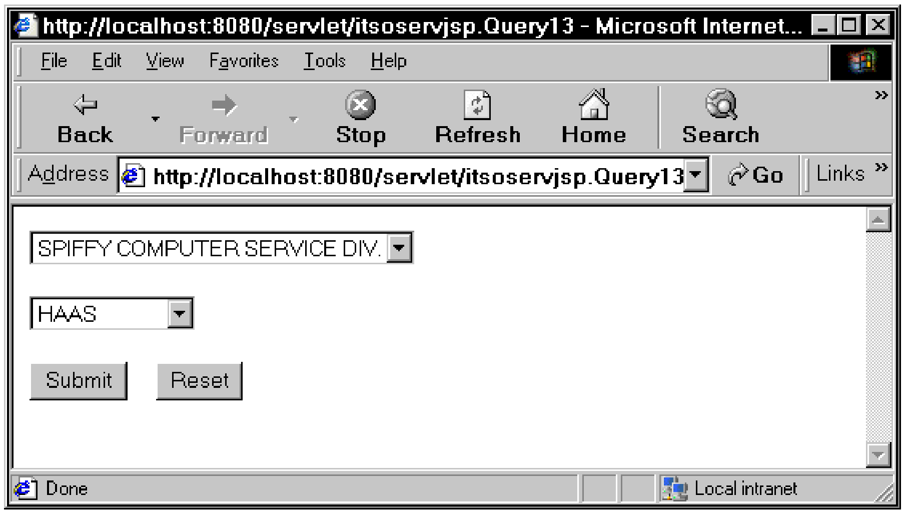
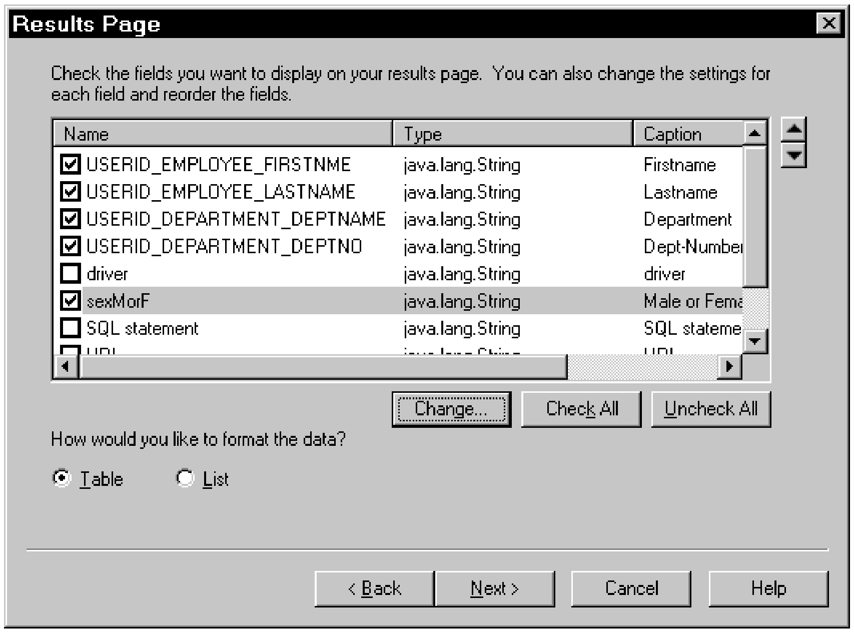

http://localhost:8080/servlet/itsoservjsp.Query13 - Microsoft Internet...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/servlet/itsoservjsp.Query13
Go Links
SPIFFY COMPUTER SERVICE DIV.
HAAS
Submit
Reset
Done
Local intranet

Figure 204. Generated results page in list format

For our example, select the result columns and the parameter. Change the caption of each result to a descriptive table heading (Figure 205).



Results Page

Check the fields you want to display on your results page. You can also change the settings for each field and reorder the fields.

Name	Type	Caption
☑ USERID_EMPLOYEE_FIRSTNME	java.lang.String	Firstname
☑ USERID_EMPLOYEE_LASTNAME	java.lang.String	Lastname
☑ USERID_DEPARTMENT_DEPTNAME	java.lang.String	Department
☑ USERID_DEPARTMENT_DEPTNO	java.lang.String	Dept-Number
☐ driver	java.lang.String	driver
☑ sexMorF	java.lang.String	Male or Fema
☐ SQL statement	java.lang.String	SQL stateme

How would you like to format the data?

[✓ Table] List

Figure 205. Database Wizard: results page

For *Select* queries, data from the returned result set is displayed. For *Insert* statements, the inserted data is displayed using the default format, which is not configurable from the wizard.

By selecting the *num affected rows* field, the number of rows affected by the *Insert* or *Update* statement is displayed in the results page. This feature may be useful during development and testing phases, where an *Update* statement may effect many rows in the database.

Standard Error page

If your Web site uses a standard error reporting page, you can specify it in this step, or you can have the wizard generate a basic page for you containing default error text. You can later edit this text in WebSphere Page Designer.

If you have an existing error page, you must know the path and file name of the page, as you cannot browse for it. The specified path is relative to the document root.

No Data page

Similar to the Standard Error page, you can display a different page if the result set from your query does not return any data.

Methods page

The Methods page enables you to select the methods of the JavaBean generated by the wizard that are to be executed. The only method presented here will be the *execute* method.

Session page

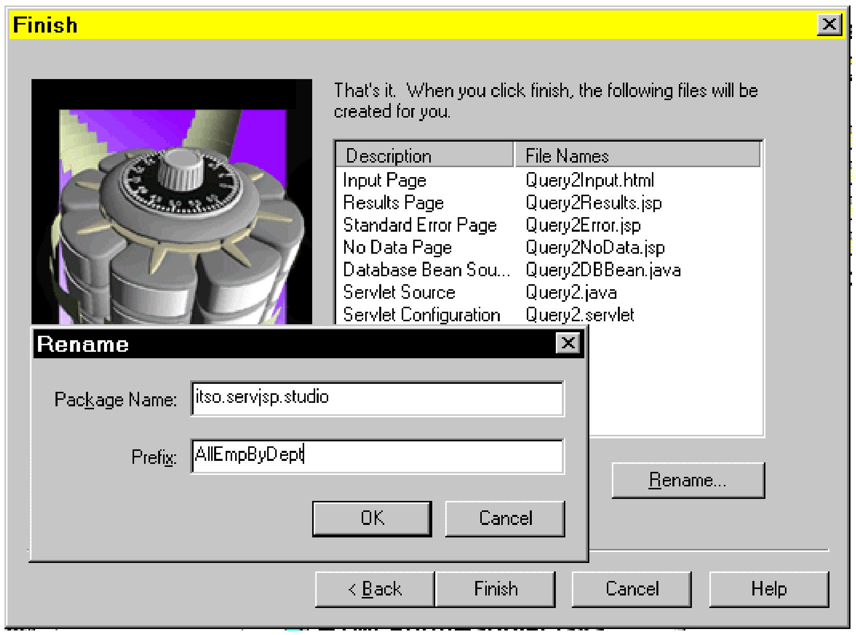
The Session page allows you to specify the scope of the bean generated by the wizard.

- ❑If you want to access the data access bean generated by the wizard from another page, select *Yes, store it in the user's session*. With this option selected, the wizard creates the bean using the `jsp:useBean` tag (if JSP 1.0 is selected) and will set its *scope* attribute to *session*.
- ❑If you do not select this option, the *scope* attribute will be set to *request*. This means that access to the bean is limited to the current page request only, and is not visible to any other pages, as each JSP page invocation is a separate request.

You should overwrite the generated name of the bean, for example, *allEmpByDeptBean*.

Finish

On the last page you are prompted with a list containing the files to be generated. You can change the default prefix for the files be selecting the *Rename* button to display the *Rename* dialog (Figure 206). You can also rename the default package name for the generated classes.



Finish

That's it. When you click finish, the following files will be created for you.

Description	File Names
Input Page	Query2Input.html
Results Page	Query2Results.jsp
Standard Error Page	Query2Error.jsp
No Data Page	Query2NoData.jsp
Database Bean Sou...	Query2DBBean.java
Servlet Source	Query2.java
Servlet Configuration	Query2.servlet

Rename

Package Name: itso.servjsp.studio

Prefix: AllEmpByDept

Figure 206. Database Wizard: tailor generated files

Click on *Finish* to generate and compile the files. The HTML and JSP files are generated into the folder you selected when the wizard was started. The servlet files are generated into the package structure specified.

Database Wizard generated code

Figure 207 shows the Studio Workbench after generation. The HTML and JSP files are in the *sql* folder. The Java files and the compiled class files are in *servlet\itso\servjsp\studio*.

There is one additional file, *AllEmpByDept.servlet*. This is the servlet configuration file required for servlets that are subclasses of *PageListServlet*. If you inspect this file with an editor, you can see all the JSP output pages listed, and also the database connection information, as initialization parameters.

You may want to change the generated DataSource value to one that you have defined in WebSphere Application Server:

```
old: <init-parameter value="jdbc/jdbcdb2sample" name="dataSourceName"/>
new: <init-parameter value="sampledb" name="dataSourceName"/>
```

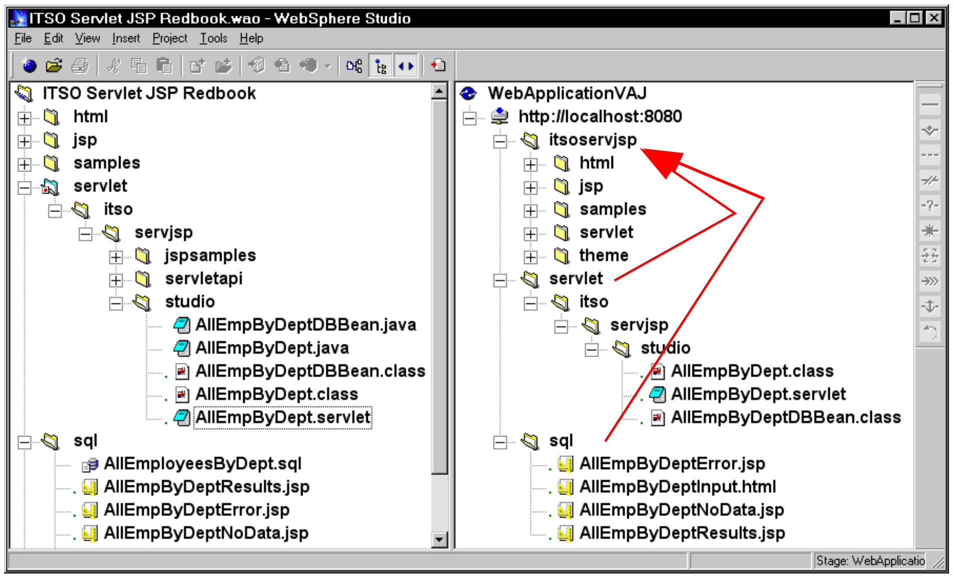


Figure 207. Database Wizard: generated files

Note. In the publishing views for Web applications, the new folders are placed at the root level and must be manually moved to the Web application folder *itsoservjsp*. Otherwise, the files will be published to the wrong directories. You have to perform this step for the *WebApplicationVAJ* and *WebApplication* publishing views.

Relationships

Select the Relations view (*View -> Relations*) and look at the diagrams for the input page, the servlet, and the result JSPs. These diagrams show you how the files are connected (Figure 208).

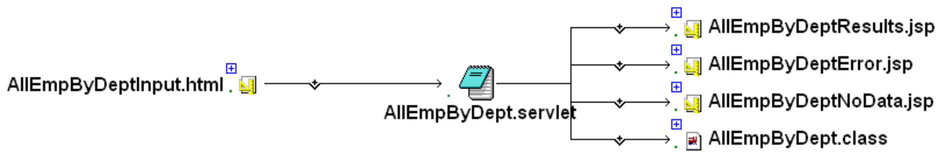


Figure 208. Database Wizard: generated relations

Run the generated application

You can run the generated code without tailoring the HTML pages and JSPs. Use either the VisualAge for Java Test Environment or WebSphere Application Server. Publish the project to the selected publishing stage.

Test in VisualAge for Java

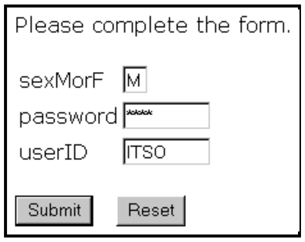
Perform these steps:

❑Prepare the SERunner or the ServletEngine class path.

❑Start the WebSphere Test Environment.

❑Launch a Web browser from the Studio Workbench by selecting *Preview File With -> [browser]* for the AllEmpByDeptInput.html file. The URL `http://localhost:8080/sql/AllEmpByDeptInput.html` is sent to the server.

❑Fill the form with suitable data:


Please complete the form.

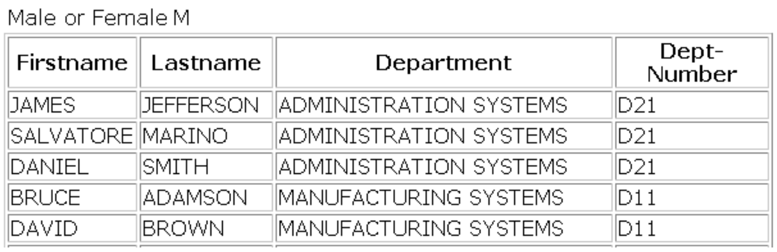
sexMorF	M
password	****
userID	ITSO

❑Click *Submit* to invoke the servlet. Our first test ended in the Debugger when executing the compiled result JSP with error message:

```
The type named itso.servjsp.studio.AllEmpByDeptDBBean is not defined
```

Import the source of the *AllEmpByDeptDBBean* class into the VisualAge for Java Workbench to resolved the syntax error in the JSP.

❑Test again, and the list of employees should appear in the browser:


Male or Female M

Firstname	Lastname	Department	Dept-Number
JAMES	JEFFERSON	ADMINISTRATION SYSTEMS	D21
SALVATORE	MARINO	ADMINISTRATION SYSTEMS	D21
DANIEL	SMITH	ADMINISTRATION SYSTEMS	D21
BRUCE	ADAMSON	MANUFACTURING SYSTEMS	D11
DAVID	BROWN	MANUFACTURING SYSTEMS	D11

....abbreviated....

Enhance the application

The generated form and JSP contain only the necessary fields, but no headings and other nice features.

You can use the Page Designer to enhance the *AllEmpByDeptInput.html* input page. Publish the changed file for testing while it is checked-out. This enables you to keep the master file until the test is successful. Then you can check-in the file to replace the master copy.

Understanding the result JSP

Open the Page Designer for the *AllEmpByDeptResult.jsp* file (Figure 209).

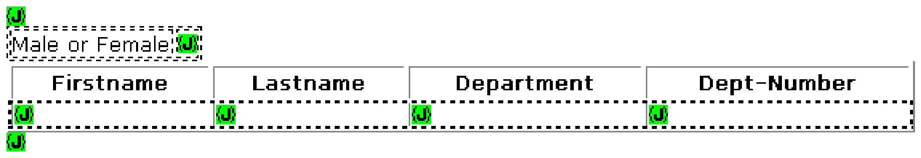


Figure 209. Database Wizard: result JSP in Page Designer

Most of the green (J) markers refer to the generated bean. Investigate the (J) markers by double-clicking:

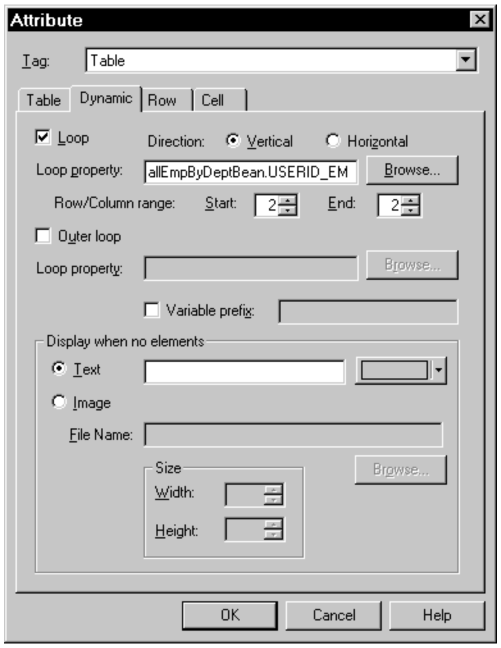
- ❑The first marker declares the AllEmpByDeptDBBean bean, similar to Figure 183 on page 244.
- ❑The second marker is the *sexMorF* property of the bean.
- ❑The four markers in the table retrieve the SQL column value properties of the bean, for example, USERID_EMPLOYEE_FIRSTNME().

  Note the parenthesis in the property. The SQL statement retrieves multiple rows, and the parentheses indicate that it is a repeating property.
- ❑The last marker is a JSP scriptlet to close the SQL result set; it contains the code `allEmpByDeptBean.closeResultSet();`

The table displays multiple rows. So where is the loop? Here is how this works:

- ❑Select the table by clicking on the outside border of the table (a pink rectangle should surround the table).
- ❑Double-click, or *Edit -> Attributes*, and the Attributes dialog appears. Select the *Dynamic* page (Figure 210). The *Loop* check box is selected, and one of the column properties of the bean is the loop property.

- ❑The setup of a loop property generates a Java *for* loop in the JSP.



[✓] Loop   Direction: [● Vertical] ○ Horizontal
Loop property: allEmpByDeptBean.USERID_EM
Row/Column range: Start: 2 End: 2
[ ] Outer loop
Loop property:
[ ] Variable prefix:
Display when no elements
[● Text] ○ Image
File Name:
Size
Width:
Height:

Figure 210. Page Designer: table loop

View the JSP source

Click on the *HTML Source* tab in the Page Designer to analyze the JSP source code. An extract of the code is shown in Figure 211. Notice:

- ❑*<jsp:usebean>* tag to declare the bean.
- ❑Each table is preceded by a *<!--METADATA>* tag that contains the reference bean properties. From this specification, the actual code is generated.
- ❑The repeating column values are retrieved into temporary variables using a *for* loop with an index variable, and the values are placed into the table using a JSP expression:

```
_p0 = allEmpByDeptBean.getUSERID_EMPLOYEE_FIRSTNME(_i0);
<TD><%= _p0 %> </TD>
```

- ❑Before the table is started, the loop property is checked with index value 0. If no data was retrieved the table is bypassed.
- ❑The *ArrayIndexOutOfBoundsException* triggers the end of the loop. You can see this exception being fired in the *AllEmpByDeptDBBean* Java code.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
 <!-- This file was generated by IBM WebSphere Studio 3.0.2 using
E:\WebSphere\Studio\BIN\GenerationStyleSheets\AppServerV3\JSP1.0\WebSphere\Pages.xsl
-->
 ......
<BODY>
<jsp:useBean id="allEmpByDeptBean" type="itso.servjsp.studio.AllEmpByDeptDBBean"
scope="request"> </jsp:useBean>
<TABLE border="0">
<TR> <TD>Male or Female</TD>
<TD>
 <!--METADATA ......<WSPX:PROPERTY property="allEmpByDeptBean.sexMorF"> -->
        <%=allEmpByDeptBean.getSexMorF() %>
......</TABLE>
<!--METADATA type="DynamicData" startspan
<TABLE border="1" width="567" dynamicelement
innerloopproperty="allEmpByDeptBean.USERID_EMPLOYEE_FIRSTNME()"
innerloopdirection="vertical" innerloopstartindex="1" innerloopendindex="1">
<TR><TH>Firstname</TH><TH>Lastname</TH><TH>Department</TH><TH>Dept-Number</TH></TR>
 <TR><TD><WSPX:PROPERTY property="allEmpByDeptBean.USERID_EMPLOYEE_FIRSTNME()"></TD>
 ...... </TR> </TABLE>-->
<%
try {
  java.lang.String _p0 = allEmpByDeptBean.getUSERID_EMPLOYEE_FIRSTNME(0); // throws an
exception if empty.
  java.lang.String _p0_0 = allEmpByDeptBean.getUSERID_DEPARTMENT_DEPTNAME(0);
  ...... %>
  <TABLE border="1" width="567">
    <TBODY>
      <TR><TH>Firstname</TH><TH>Lastname</TH><TH>Department</TH>...... </TR>
<% for (int _i0 = 0; ; ) { %>
        <TR> <TD><%= _p0 %> </TD> <TD><%= _p0_2 %> </TD> ...... </TR><%
        _i0++;
        try {
          _p0 = allEmpByDeptBean.getUSERID_EMPLOYEE_FIRSTNME(_i0);
          _p0_0 = allEmpByDeptBean.getUSERID_DEPARTMENT_DEPTNAME(_i0);
          ...... }
        catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
          break;
        }
      } %>
    </TBODY>
  </TABLE><%
}
catch (java.lang.ArrayIndexOutOfBoundsException _e0) { } %>
<!--METADATA type="DynamicData" endspan-->
<%allEmpByDeptBean.closeResultSet();%>
</BODY>
</HTML>
```

Figure 211. Database Wizard: generated result JSP source (extract)

You can use the Page Designer to enhance the result page with a heading and other HTML features.

JavaBean Wizard

The JavaBean Wizard helps you create Web pages based on properties of a JavaBean. The JavaBean Wizard interrogates the specified Bean object and steps you through the process of creating pages to display and update the properties of the Bean. In addition to normal JavaBean objects, the wizard supports *command* and *navigator* beans developed in Visual Age for Java, and *access* beans created in the VisualAge for Java EJB development environment.

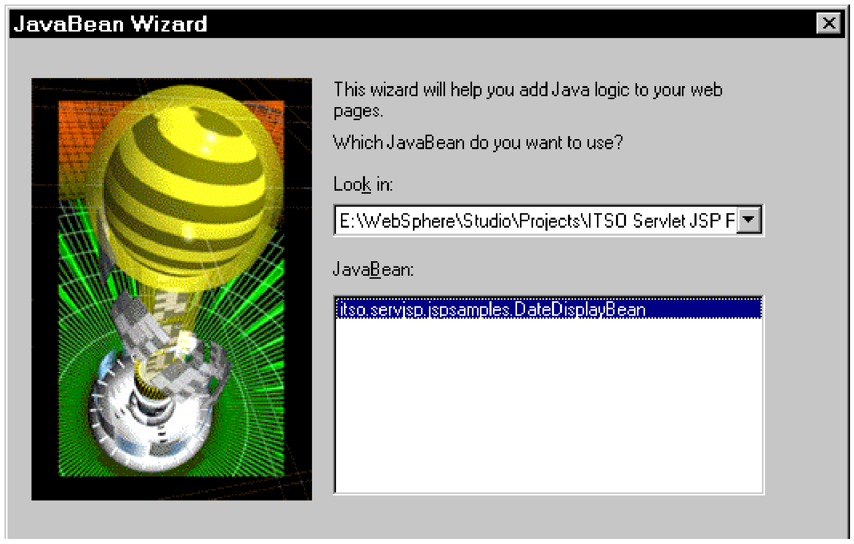
Before using the JavaBean Wizard

To invoke and use the JavaBean Wizard, you must have at least one JavaBean in your project, and the folder structure containing the JavaBean you want to use must match the package name of the Bean.

For example, the `DateDisplayBean` used in this section must be placed in the path `itso\servjsp\jspsamples\DateDisplayBean` underneath the servlets node in the WebSphere Studio project view. You have to manually create this folder structure.

Run the JavaBean Wizard

Start the wizard from the *Tools* menu and the wizard welcome dialog appears (Figure 212).



This wizard will help you add Java logic to your web pages.

Which JavaBean do you want to use?

itso.servjsp.jspsamples.DateDisplayBean

Figure 212. JavaBean Wizard: select a bean

JavaBean Wizard

In the first page of the wizard, you must specify the JavaBean that the wizard will use. Select the bean from the drop-down list of available beans, and the full package and name of the bean is displayed (Figure 212).

Web pages

The Web pages dialog allows you to specify the pages produced by the wizard. The function of each page is described in Table 18.

Table 18. Database Wizard generated pages

Page Type	**Description**
Input page	A Web page containing user input fields to capture data from the user. The data entered in this form will be used to update the properties of the bean. The form field data is passed as URL parameters to the servlet generated by the wizard.
Results page	A page containing the properties of the JavaBean. You can specify which properties are displayed in a later step in the wizard.
Error page	A page containing error text displayed when an unexpected error occurs. An error may be triggered by the bean not being found or an error in the bean syntax.

Input page

Here you specify which fields are included in the HTML input page generated by the wizard. The wizard interrogates the Bean object and lists the properties exposed by the JavaBean.

In our example, only one property (*counter*) has a set method, and it is displayed in the list of properties. The semantics of the input field and its associated caption (*Enter the counter value:*) can be changed by highlighting the property and clicking the *Change* button (similar to Figure 203 on page 267).

Results page

The results JSP is displayed following successful invocation and processing of the input page. You can display the data entered by the user in the input page and all the bean properties. In our example, select the *counter* and the *dateString* properties and change the captions for the output.

Standard Error page

You can optionally specify a JSP page that is displayed if errors occur. For more information see “Standard Error page” on page 269).

Methods page

The *Methods* dialog enables you to specify additional methods that are executed in the servlet generated by the wizard. By default, no additional methods are selected. If you specify additional methods, they will be called from the *performTask* method of the generated servlet.

The standard methods of the servlet produced by the wizard are executed as follows:

```
doPost/doGet -> performTask -> [set propertys] -> [call additional methods]
```

Figure 213 shows the servlet code produced when the method *toString* is specified.

```
public void performTask(HttpServletRequest request, HttpServletResponse response)
  {
    try
    {
     // instantiate the beans and store them so they can be accessed by the called page
      itso.servjsp.jspsamples.DateDisplayBean dateDisplayBean = null;
      dateDisplayBean = (itso.servjsp.jspsamples.DateDisplayBean)
                           java.beans.Beans.instantiate(getClass().getClassLoader(),
                           "itso.servjsp.jspsamples.DateDisplayBean");

      setRequestAttribute("dateDisplayBean", dateDisplayBean, request);

      // Initialize the bean counter property from the parameters
        dateDisplayBean.setCounter(Integer.valueOf(
              getParameter(request, "counter", true, true, true, null)).intValue());

      // Call the toString action on the bean.
      dateDisplayBean.toString();

      // Call the output page. If the output page is not passed
      // as part of the URL, the default page is called.
      callPage(getPageNameFromRequest(request), request, response);
    }
    catch (Throwable theException)
    {
      // uncomment the following line when unexpected exceptions are occuring
                                                 to aid in debugging the problem
      // theException.printStackTrace();
      handleError(request, response, theException);
    }
  }
```

Figure 213. Code snippet demonstrating calling additional methods

When clicking *Next*, we got an error box that the package name contains invalid characters. This is because the default package name is the project name, and our project includes blanks.

Finish

The last step of the wizard displays the list of generated files. Click *Rename* to specify the package name (*itso.servjsp.studio*) and the prefix (*DateBeanWiz*). This is similar to the Database Wizard (Figure 206 on page 270).

We got an error message from code generation:

```
E:\WebSphere\Studio\check_out\ITSO Servlet JSP Redbook\servlet\
   itso\servjsp\studio\DataBeanWiz.java:20:
   Class ITSOServletJSPRedbook.DateDisplayBean not found in import.
import ITSOServletJSPRedbook.DateDisplayBean;
```

The generated servlet uses the wrong package for the DateDisplayBean!

Fix the broken code

Edit the DateBeanWiz.java source code and replace all occurrences of *ITSOServletJSPRedbook* with *itso.servjsp.jspsamples*. Recompile the source, then check-in the changed files.

You also have to edit the generated JSP files and change the package name of the DateDisplayBean.

Organize the folders

The generated HTML and JSP files are in the root folder. Move these files to the HTML and JSP folders.

Check the publishing views. For publishing to a Web application, you must move all folders under the *itsoservjsp* folder.

Tailor the input form and the output JSP

Optionally, you can use the Page Designer to edit the input and result page and improve their appearance.

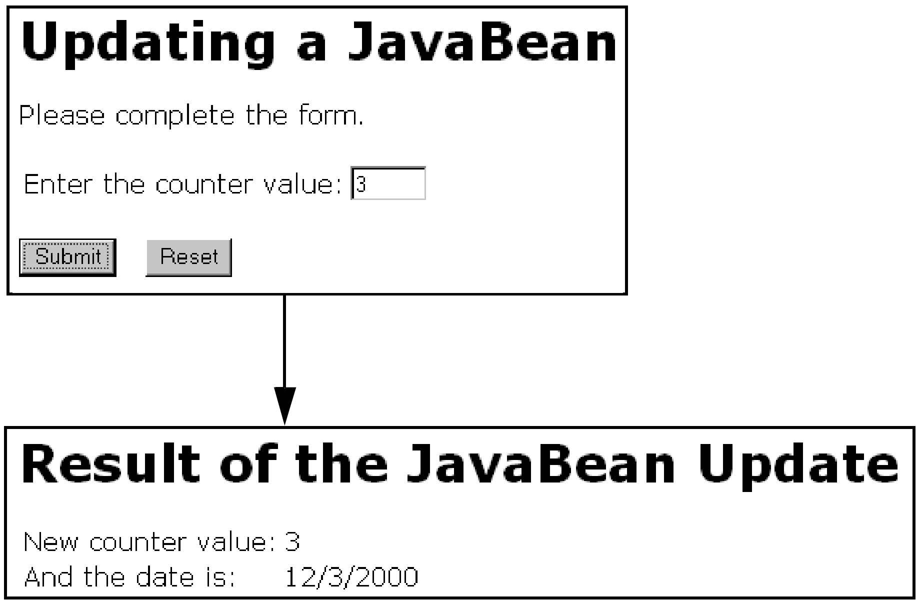
Test the JavaBean Wizard code

Before testing, import the DateDisplayBean into the Workbench, otherwise you will get syntax errors in the JSPs.

Check that the SERunner (or ServletEngine) class path includes the servlet directory:

```
D:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\
                                   hosts\default_host\...webapp....\servlets\;
```

Publish the files, and start the test from the DateBeanWizInput.html file. The browser should display output as shown in Figure 214.



Updating a JavaBean

Please complete the form.

Enter the counter value: 3

Result of the JavaBean Update

New counter value: 3
And the date is: 12/3/2000

Figure 214. JavaBean Wizard: test run

JavaBean Wizard — what for?

This example does not make much sense, because the JavaBean is not persistent and does not perform any processing.

However, just imagine that the JavaBean is an access bean for an Enterprise JavaBean, or a command bean that connects to a back-end system, and you will understand the power of the concept.

Developing an application in WebSphere Studio

This section discusses the necessary steps to develop an extended application using the DEPARTMENT, EMPLOYEE, and EMP_PHOTO tables of the DB2 sample database. The example was developed using WebSphere Studio and demonstrates the following scenario:

- ❑Display a form where the user can enter a department number.
- ❑Display a list of employees in that department.
- ❑For each employee, check if a GIF photo is available in the EMP_PHOTO table. If so, provide an HTML reference link for the photo.
- ❑When a photo link is clicked, display the photo in the browser.

In the example, we used the SQL Wizard and Database Wizard to generate the input and output pages, the servlets, and the data access beans. We also developed a stand-alone JSP to display an employee photo using one of the data access beans.

We had to make a few changes to the generated code to deal with the photo BLOB and to implement the optional link to the employee photo.

The steps are described in sequence. Some of the details of the SQL and Database Wizards are omitted because they can be performed in exactly the same way as described in the previous sections about the wizards.

Create the SQL statement for the employees of a department

Define a new folder called *photo* for this application.

Start the SQL Wizard and create an SQL statement called *EmpInDept* with the select statement shown in Figure 215.

```
SELECT DISTINCT USERID.EMPLOYEE.EMPNO, USERID.EMPLOYEE.LASTNAME,
  USERID.EMPLOYEE.JOB, USERID.EMPLOYEE.SEX, USERID.EMPLOYEE.SALARY,
  USERID.EMP_PHOTO.PHOTO_FORMAT
 FROM USERID.EMPLOYEE LEFT OUTER JOIN USERID.EMP_PHOTO ON
                        ( USERID.EMPLOYEE.EMPNO = USERID.EMP_PHOTO.EMPNO )
WHERE ( ( USERID.EMPLOYEE.WORKDEPT = ? ) AND
        ( ( USERID.EMP_PHOTO.PHOTO_FORMAT = 'gif' ) OR
          ( USERID.EMP_PHOTO.PHOTO_FORMAT is null )    )  )
ORDER BY USERID.EMPLOYEE.EMPNO
```

Figure 215. SQL statement for employees in a department

This statement returns all the employees in a given department. The photo format is returned if it is GIF or null. We must use an outer join, because there are no photographs for most employees.

Note: Your statement may require a right outer join, such as:

```
FROM USERID.EMP_PHOTO RIGHT OUTER JOIN USERID.EMPLOYEE ON
                       ( USERID.EMP_PHOTO.EMPNO = USERID.EMPLOYEE.EMPNO )
```

Here are the steps in the SQL Wizard:

❑Name the statement *EmpInDept* and logon as ITSO/itso.

❑Specify a *Select Unique* and select the EMPLOYEE and EMP_PHOTO tables.

❑Join the two tables on the EMPNO column. Click on *Options* and select the *left outer join*.

❑Select the columns EMPNO, LASTNAME, JOB, SEX, SALARY (from EMPLOYEE), and PHOTO_FORMAT (from EMP_PHOTO).

❑First condition: WORKDEPT is exactly equal to *deptnum* (a parameter).

❑Second condition (AND): PHOTO_FORMAT is exactly equal to *gif* (a constant)

❑Third condition (OR): PHOTO_FORMAT is blank (this means null).

❑Sort ascending by EMPNO.

❑Check that the SQL statement matches the statement in Figure 215.

Note one difference: **The wizard cannot generate the extra set of parentheses around the OR conditions**. You have to fix this later in the generated code!

❑Finish to save the statement.

Create the SQL statement for the employee photo

Start the SQL Wizard and create an SQL statement called *EmpPhoto* with the select statement shown in Figure 216.

```
SELECT USERID.EMP_PHOTO.PICTURE
  FROM USERID.EMP_PHOTO
 WHERE ( ( FUNDY.EMP_PHOTO.EMPNO = ? ) AND
         ( FUNDY.EMP_PHOTO.PHOTO_FORMAT = 'gif' ) )
```

Figure 216. SQL statement for employee photos

This statement returns the employee picture for a given employee number.

Here are the steps in the SQL Wizard:

❑Name the statement *EmpPhoto* and logon as ITSO/itso.
❑Specify a *Select* and select the EMP_PHOTO table.
❑Select the PICTURE column.
❑First condition: EMPNO is exactly equal to *empno* (a variable).
❑Second condition (AND): PHOTO_FORMAT is exactly equal to *gif* (a constant).
❑Check the SQL statement. It should match Figure 216.
❑Click *Finish* to save the statement.

Generate the code for the employees in a department

Here are the steps in the Database Wizard:

❑Start the Database Wizard and select the *EmpInDept* SQL statement.
❑Web pages: Select all four pages to be generated.
❑Input page: Select the *deptnum* field and change the caption to some descriptive text (Enter a department number:); change the length to 3.
❑Results page: Select the table columns and the *deptnum* field. Change the captions to appropriate table headings.
❑Session: Select *No* for session and name the bean *empdeptBean*.
❑Finish: click *Rename* and set the package to *itso.servjsp.photo* and the prefix as *Empdept*. Click *Finish* to generate the code.

Generate the code for the employee photo

Here are the steps in the Database Wizard:

❑Start the Database Wizard and select the *EmpPhoto* SQL statement.
❑Web pages: Select all four pages to be generated.
❑Input page: Select the *empno* field and change the caption to some descriptive text (Enter an employee number:); change the length to 6.
❑Results page: Select only the table column and change the caption to appropriate table heading (Photo).
❑Session: Select *No* for session and name the bean *photoBean*.
❑Finish: Click *Rename* and set the package to *itso.servjsp.photo* and the prefix as *photo*. Click *Finish* to generate the code.

Change the generated DataSource

Edit the generated servlet configuration files (Empdept.servlet and photo.servlet) and change the DataSource from *jdbc/jdbcdb2sample* to *sampledb*, which is the DataSource we defined in WebSphere Application Server (see “Creating a DataSource” on page 146).

Fixing the problems

The generated code has two problems. The first SQL statement has a missing set of parentheses for the OR conditions, and the second SQL statement uses a wrong data type for the picture BLOB.

Changing the SQL statement

Edit the EmpDeptDBBean.java file and add the extra parenthesis around the OR condition. Save the file and compile it. Check-in the Java and class file.

Changing the Java data type for the picture

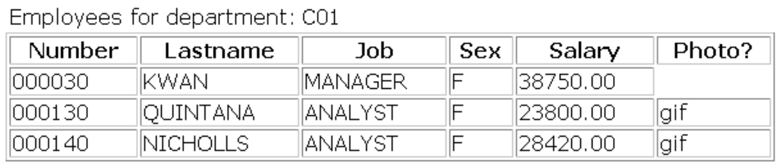
The data type of the BLOB is generated as *java.lang.Byte*, instead of *byte[]*. Edit the photoDBBean.java file, search for java.lang.Byte and replace the 3 occurrences with byte[]. Save the file and compile it. Check-in the Java and class file.

The *photoResults.jsp* file uses the bean and also has the java.lang.Byte data type. Open the bean with the Page Designer and check the JSP source code. It should now pick up the new data type (byte[]) from the bean. Save the file.

Testing in VisualAge for Java

Publish all the new files, basically, the *photo* and the *itso.servjsp.photo* folders. Import the two beans into the Workbench, EmpdeptDBBean and photoDBBean.

Start the WebSphere Test Environment and launch the browser for the *EmpdeptInput.html* file. Enter C01 or D11 as department number (the other departments have no pictures). A sample output is shown in Figure 217.



Employees for department: C01

Number	Lastname	Job	Sex	Salary	Photo?
000030	KWAN	MANAGER	F	38750.00	
000130	QUINTANA	ANALYST	F	23800.00	gif
000140	NICHOLLS	ANALYST	F	28420.00	gif

Figure 217. Employees in department test run

Now launch the browser for the *photoInput.html* file and enter 000130 as employee number. A sample output is shown in Figure 218.

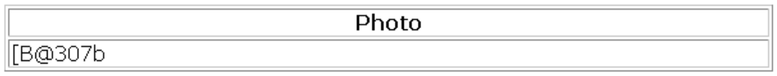


Photo
[B@307b

Figure 218. Employee photo test run

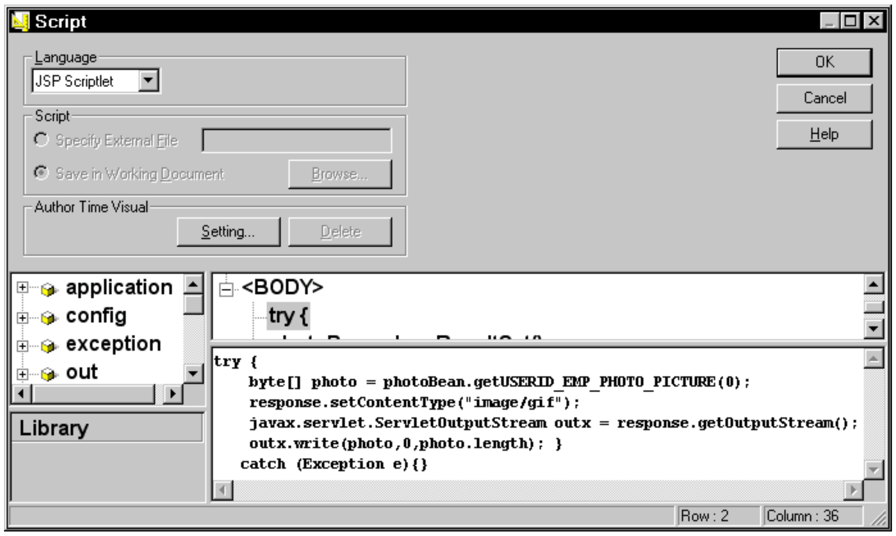
At this point the code to display the real GIF picture is not there yet, and the link between the first servlet and the second servlet is missing too.

Displaying a picture

How do we display the picture? We have to create a different kind of output, instead of *text/html* we use *image/gif* and write the byte stream to the response object.

Open the *photoResult.jsp* in the Page Designer. Delete the (J) marker in the table, and add a scriptlet instead (*Insert -> JSP Tags -> Scriptlet*). Enter the code into the box (Figure 219).

```
try {
   byte[] photo = photoBean.getUSERID_EMP_PHOTO_PICTURE(0);
   response.setContentType("image/gif");
   javax.servlet.ServletOutputStream outx = response.getOutputStream();
   outx.write(photo,0,photo.length); }
  catch (Exception e){}
```



Language: JSP Scriptlet

```
try {
   byte[] photo = photoBean.getUSERID_EMP_PHOTO_PICTURE(0);
   response.setContentType("image/gif");
   javax.servlet.ServletOutputStream outx = response.getOutputStream();
   outx.write(photo,0,photo.length); }
  catch (Exception e){}
```

Row : 2   Column : 36

Figure 219. Employee photo scriptlet

Next, open the table itself and deselect the *loop* property on the *Dynamic* page. There is only one picture per employee.

Test with the modified JSP and the picture should be displayed by the browser. Note that the table itself is not shown, just the picture.

Linking the servlets

The changes needed to link from the department listing to the photo servlet are in the *EmpdeptResults.jsp* output page. When the value in the photo column is *gif*, we have to add HTML tags with a reference to the photo servlet into that column.

What we want to construct is a conditional link:

```
if photo_format = gif then
   <a href="itso.servjsp.photo.photo">Display</A>
```

Edit the *EmpdeptResults.jsp* with the Page Designer.

❑Check the source code. The variable used in the table loop is *_i0*. We have to use the variable in our test of the picture format.

❑Delete the (J) marker in the photo column. We do not display the photo format. Instead we build the HTML reference.

❑Add this scriptlet into the table column:

```
if ( empdeptBean.getUSERID_EMP_PHOTO_PHOTO_FORMAT(_i0) != null ) {
```

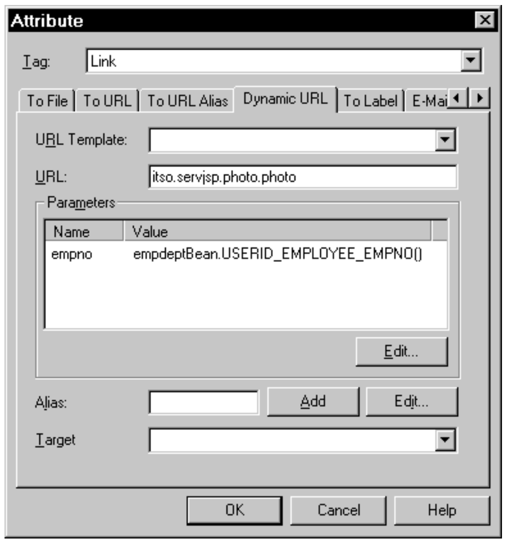
Note that you can drag properties from the left pane into the code pane to build the code, then you replace the index with _i0.

❑Add a second scriptlet (to end the *if* statement) with the code: }

❑Between the two scriptlets, add a link (*Insert -> Link*), and an Attribute dialog opens (Figure 220).

- On the *Dynamic URL* page, enter *itso.servjsp.photo.photo*, the name of the target servlet, as the URL.
- Click on *Edit* in the Parameters pane. This opens the URL Parameter Editor dialog. Enter *empno* as name. For the value select *Specify by property* and click on *Browse*.
- In the Bean Property Selection dialog, select the EMPNO property of the *empdeptBean* and click OK.
- In the URL Parameter Editor, click *Add* to add this parameter to the list. Close the dialog.

Figure 220 shows the completed Attribute dialog. Close the dialog.


Attribute

Tag: Link

To File | To URL | To URL Alias | Dynamic URL | To Label | E-Mai

URL Template:

URL: itso.servjsp.photo.photo

Parameters

Name	Value
empno	empdeptBean.USERID_EMPLOYEE_EMPNO()

Figure 220. Dynamic HTML link

- ❑This dialog enters the link into the table. The text for the link is taken directly from the URL and reads as `itso.servjsp.phot.photo`.
- ❑You can overtype the text with `Display`, which will be the link.

JSP compile problem

If you save and publish this code, the JSP does not compile in JSP 1.0. The code that is inserted into the METADATA tag for the table produces a compile error.

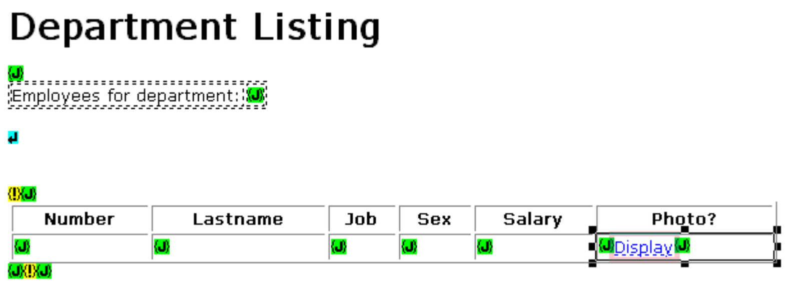
To fix that compile error, switch to the source view. Find the METADATA tag that starts like:

```
<!--METADATA type="DynamicData" startspan
<TABLE border="1" width="600" dynamicelement
innerloopproperty="empdeptBean.USERID_EMPLOYEE_EMPNO()"
innerloopdirection="vertical" innerloopstartindex="1" innerloopendindex="1">
.......
</TABLE>
-->
```

Change this code to be a JSP comment so that nothing inside is compiled:

```
<%--METADATA .............................
--%>
```

Save the completed JSP (Figure 221), publish it, and test again.

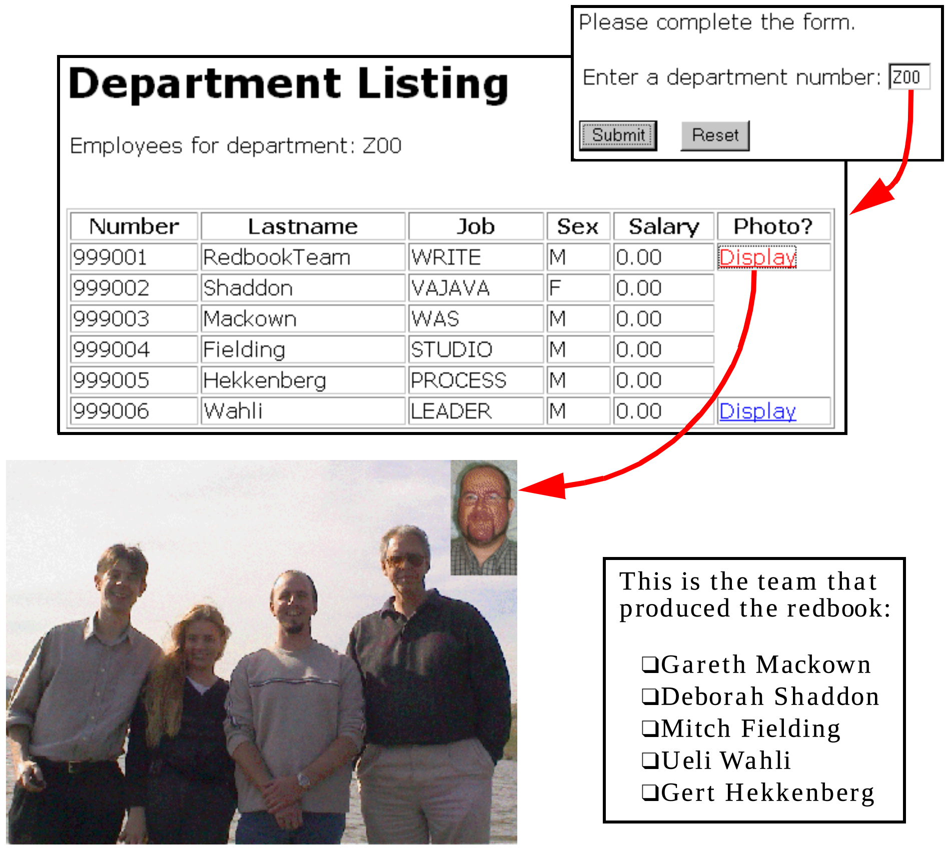


Number	Lastname	Job	Sex	Salary	Photo?
					Display

Figure 221. Completed employee in department JSP

Run the application

The complete application is shown in Figure 222.



Employees for department: Z00

Number	Lastname	Job	Sex	Salary	Photo?
999001	RedbookTeam	WRITE	M	0.00	Display
999002	Shaddon	VAJAVA	F	0.00	
999003	Mackown	WAS	M	0.00	
999004	Fielding	STUDIO	M	0.00	
999005	Hekkenberg	PROCESS	M	0.00	
999006	Wahli	LEADER	M	0.00	Display

This is the team that produced the redbook:

- ❑Gareth Mackown
- ❑Deborah Shaddon
- ❑Mitch Fielding
- ❑Ueli Wahli
- ❑Gert Hekkenberg

Figure 222. Complete application flow

Problems

Not everything worked fine with this level of the WebSphere Studio product. We have already touched on some of the problems we encountered during the previous sections. In the sections below, we discuss these problems.

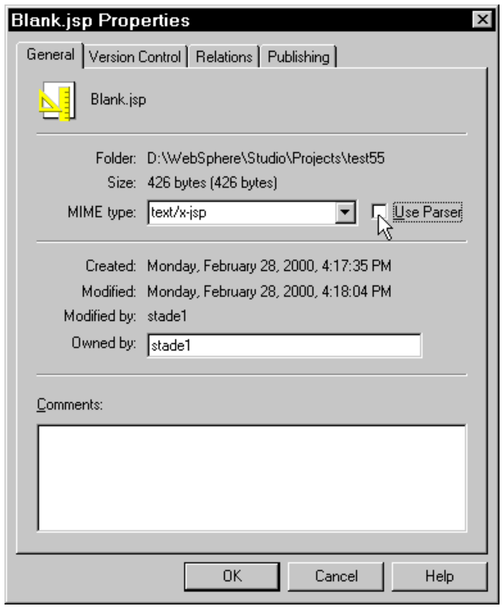
Resolving parsing problems

Occasionally, the parsing engine used by WebSphere Studio updates the links between parts incorrectly.

For example, in WebSphere Studio 3.02, a scriptlet embedded in the ACTION attribute of a FORM declaration gets incorrectly prefixed with a leading "/" when the file is published:

```
<FORM action="<%= myObject.myProperty %>" method="POST"></FORM>      <== original
<FORM action="/<%= myObject.myProperty %>" method="POST"></FORM>     <== after parse
```

To overcome any parsing problems such as this, select the file in WebSphere Studio and *Edit -> Properties* (Figure 223). By deselecting the *Use Parser* check box, you can force WebSphere Studio to not change URL information in this file during the publishing operation.



Blank.jsp

Folder: D:\WebSphere\Studio\Projects\test55
Size: 426 bytes (426 bytes)
MIME type: text/x-jsp   ☐ Use Parser

Created: Monday, February 28, 2000, 4:17:35 PM
Modified: Monday, February 28, 2000, 4:18:04 PM
Modified by: stade1
Owned by: stade1

Figure 223. Toggling the Use Parser checkbox on a file

Folders in publishing stages for a Web application

When new folders are created, they are not automatically subfolders of the Web application folder in the publishing view. You have to move them manually. This is even true for the servlet folder that already exists in the Web application folder.

Refer to “Publishing to a Web application” on page 251 for directions on how to publish to a Web application.

SQL Wizard generates wrong data type for a BLOB column

The SQL Wizard generates the Java type *java.lang.Byte* instead of a byte array (*byte[]*) for a BLOB column.

Database Wizard JSP code is compiled within METADATA tag

The code to retrieve properties that is placed into a METADATA tag in the result JSP is compiled by the JSP 1.0 compiler. (It is not compiled under JSP 0.91.)

When updating such a result table with user defined code in a scriptlet, this can lead to compile errors.

A circumvention is to change the METADATA comment into a JSP comment (as discussed in “JSP compile problem” on page 287), but we have seen problems when such a JSP is updated a few times.

JavaBean Wizard generates bad code

The JavaBean Wizard generates the wrong package name if the project name contains blanks or other invalid characters. See “Finish” on page 279.

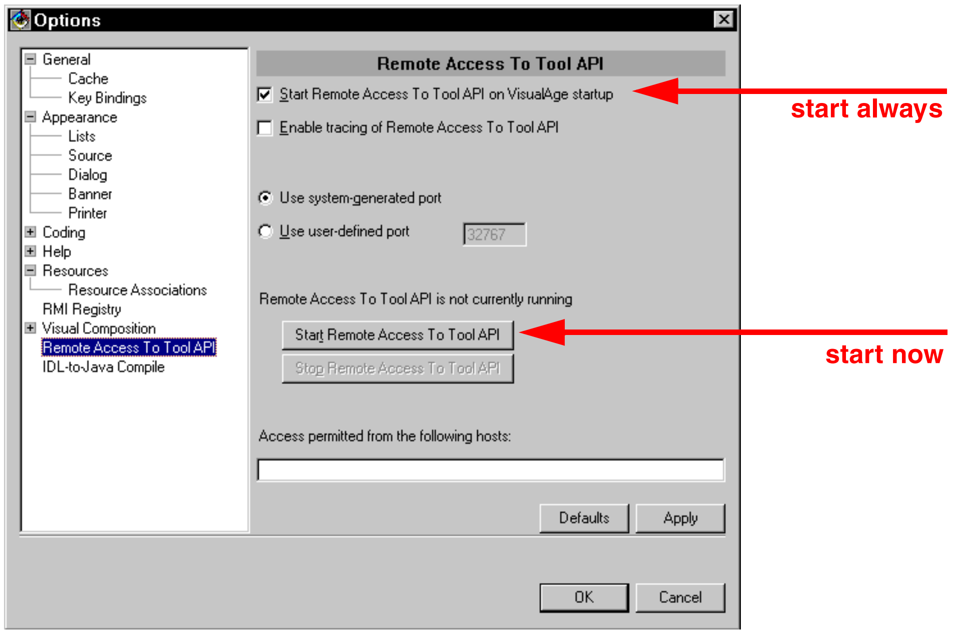
Interfacing to VisualAge for Java

WebSphere Studio provides two-way communication with Visual Age for Java, allowing you to keep your Java source files synchronized between WebSphere Studio and the VisualAge for Java repository.

You should use this feature if VisualAge for Java is your primary Java development and test tool.

Setting up the environment

Before WebSphere Studio can interface to Visual Age for Java, you must enable the *Remote Access to Tool API* feature within VisualAge for Java. Select *Window* -> *Options* to display the dialog shown in Figure 224.



[✓] Start Remote Access To Tool API on VisualAge startup
[ ] Enable tracing of Remote Access To Tool API
[● Use system-generated port] / ○ Use user-defined port: 32767
Remote Access To Tool API is not currently running
Access permitted from the following hosts:

Figure 224. Configuring Visual Age for Java for WebSphere Studio interface

If you will be using this feature regularly, ensure that the *Start Remote Access to Tool API on VisualAge startup* option is checked. To start the communication, click the *Start Remote Access to Tool API* button.

WebSphere Studio

The menu interface to VisualAge for Java is under *Project -> VisualAge for Java*, once you select a Java or class file.

If the menu items in WebSphere Studio are greyed-out, you may have installed WebSphere Studio before you installed Visual Age for Java. If so, try reinstalling WebSphere Studio *after* you have installed Visual Age for Java.

Receiving updates from Visual Age for Java

Similarly, you can update the WebSphere Studio project's files with the source updated in the Visual Age for Java projects.

Highlight the files that you want to update from Visual Age for Java. Note that you should highlight both the `.java` *and* `.class` files if you want the `.java` source file to be updated, otherwise you have to compile the source file yourself afterwards.

Select *Project -> Visual Age for Java -> Update from VisualAge* (Figure 225). Again, no visual notification is provided, so manually verify that the update was successful by checking your code.

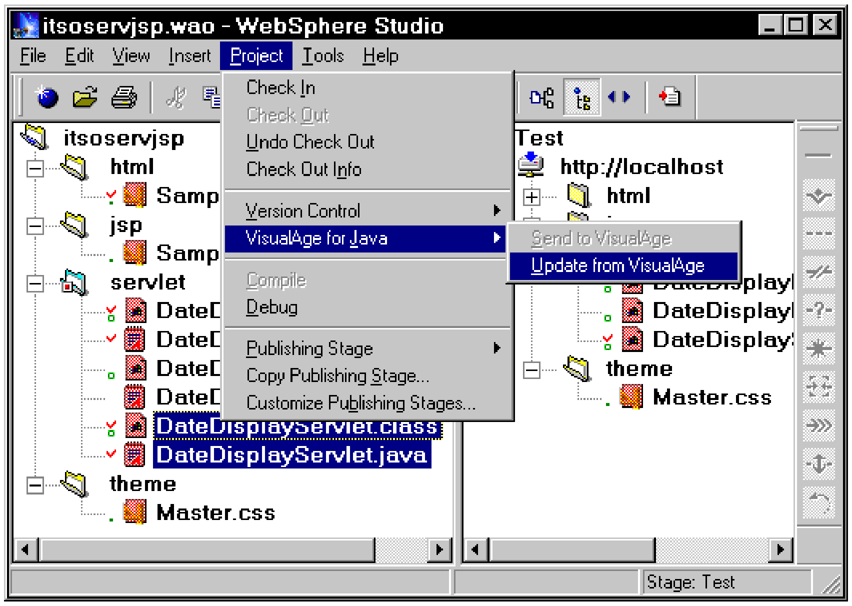


Figure 225. Updating WebSphere Studio files from Visual Age for Java

Sending updates to Visual Age for Java

If you make changes to Java source code files in WebSphere Studio, you can send these changes to Visual Age for Java.

Select *Project -> Visual Age for Java -> Send to VisualAge*. You must tell VisualAge for Java the project into which the class is inserted. Once this is done, you will not receive any further visual notification that the operation is complete; however, you can simply view the class in VisualAge for Java to check that the update operation was successful.

Using VisualAge for Java as an editor

See "Editing Studio files with VisualAge for Java" on page 390 for instructions on how to register VisualAge for Java as an editor for Java files.

Archiving

You can archive a whole project into a WebSphere Studio archive (.war) file. Select *File -> Save as Archive* and enter the name of the generated file in the dialog that follows.

An archive file contains the project structure, publishing stages, and all the files.

Opening an archive

You can restore an archive file onto a developer's machine by using the *File -> Open Archive* action. A dialog opens and you should carefully go through the three pages:

❑On the *Extract* page, either create a new project or replace the current project folders (a merge operation).

❑On the *Destination* page (only for a new project), select original locations or custom locations. If you use original locations, the disk letter must exist on your system, otherwise extract fails with an error box saying that a file could not be written (not a very helpful message). With custom locations, you can control the directory where the project will be stored.

❑On the *Options* page, select *Use archived publishing targets* if you want to preserve the publishing target locations for the publishing stages. If you do not select this check box, the publishing targets are lost, and you must update the project manually.

Working in a team

Multiple team members can use the same directory structure for their Studio project. Files (HTML, JSP, Java source) are checked-out for editing by team members. A checked-out file cannot be edited by another developer.

This support is not very comprehensive, and it only protects the files if all developers use the Studio Workbench and do not modify files outside of the Workbench.

More information and examples

Refer to Chapter 15, “Developing the PDK using WebSphere Studio” on page 383 for more information and examples of using WebSphere Studio.

9 Software Configuration Management

In this chapter we describe the challenging area of Software Configuration Management (SCM), and how it relates to WebSphere Studio and VisualAge for Java.

While the starting point of customer involvement with SCM varies, no customer can afford to ignore this area. In fact, after implementing SCM processes, the resulting improvements in IT reaction times to meet business demands could well prove to be a key success factor for being successful with e-business.

Introduction

SCM is one of the key areas that has to be addressed when developing and maintaining applications. This is not only true for managing the software configuration within your development environment, but also applies to the software configuration within the production environment.

Application architectures, methodologies, technologies, and associated tools put into a development process context potentially fail on delivering the appropriate functionality to the business if SCM processes and supporting tools are not in place.

Pressure to deliver faster and more complex applications makes it more urgent to implement SCM. At the same time, businesses that are developing and deploying applications in the e-business space may find themselves open to exposure when SCM problems occur.

Although this calls for an end-to-end (E2E) approach for SCM throughout the complete application life cycle, we will limit ourselves by addressing some SCM aspects within the scope of this book. Although an E2E approach is still advisable, addressing all aspects of SCM would be a book in itself.

We will illustrate some aspects of SCM using Rational's ClearCase product. Our choice for ClearCase is driven by the fact that ClearCase has a prominent role within IBM's SCM strategy.

First, we will start with some general thoughts on SCM.

What is Software Configuration Management?

The U.S Department of Defense, in its standard on software development, DOD-STD-2167A, defines SCM as follows:

> *Software Configuration Management is the discipline of identifying the configuration of software systems at discrete points in time for the purpose of controlling changes and maintaining traceability of changes throughout the software life cycle.*

Other definitions from IEEE or ISO are more or less the same, although the focus differs.

Over the years, two groups have expanded on the definition of SCM, with each providing a different perspective. One group, the Software Engineering Institute (SEI) at Carnegie Mellon University, has approached SCM from the process side; while the second group, the International Standards Organization (ISO), has approached it from the management side.

While it appears that these groups have worked in different areas, they have in fact addressed the two major areas needed for successfully implementing SCM. It is not only important to know WHAT to manage and HOW to manage development artifacts, but this should be put into a context, which calls for a repeatable process.

Successful software development organizations are measured by SEI maturity levels and by compliance with ISO 9000 standards. These maturity levels and standards present new challenges to development teams that are being asked to deliver software to market at faster and faster rates, while at the same time improving the quality.

In order to raise an organization's maturity level or to comply with ISO 9000 standards, tools are required. These tools must support the following related and sometimes overlapping functional areas:

❑Version management

The capability of managing versions is one of the building blocks of SCM. Being able to restore to the exact point-in-time when an application was working properly is just one example of the necessity of versioning.

❑Change management

Being able to apply changes to an existing situation in a controlled manner is important. This is especially true when applying changes to the production environment. In order to have flexibility in this respect, there is a requirement to be able to separate changes, attach them to identified work items, and create flexible baselines based upon sets of changes.

❑Change request tracking

Strongly related to change management, this means tracking change requests and associated work items over time during their life cycle. Tracking is applicable to both planned work items (new release, improvements) and unplanned work items (problems).

❑Build management

Throughout the development life cycle, there is a requirement to be able to build the software at hand in a repeatable manner. This might call for multiple platform build capabilities if your applications span multiple platforms.

- ❑Deployment

  A natural extension to build management is to have functions available that are capable of deploying the resulting artifacts to the execution environment, which can reside on multiple platforms.

- ❑Impact analysis

  From both a development and runtime perspective, there is a requirement to be able to predict and assess the impact potential changes have.

- ❑Process

  The above-mentioned functions are not independent from each other. They should work in a concerted manner. At least, the functions should be embedded into a set of procedures to follow when applying changes to IT solutions. Given the importance of this aspect, it calls for a tool that supports and enforces an SCM process while at the same time providing flexibility in this respect.

As indicated, we will only skim the surface of SCM in this book. We wanted to illustrate in this section that SCM is important, and which areas it should cover. If this summary has fueled your interest, we suggest that you read *Managing the Software Process*, by Watts S. Humphrey (see “Other resources” on page 435).

SCM for architectural pattern based development

Chapter 8 of the redbook *Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition,* SG24-5864, addresses various aspects of the application development process and associated work products that are being created throughout the application development life cycle.

We will build upon that chapter by adding some SCM aspects. In the introduction, we said that SCM processes and implementing tools should cover the whole application life cycle, covering not only the development phase but also the deployment and maintenance phases as well (Figure 226).

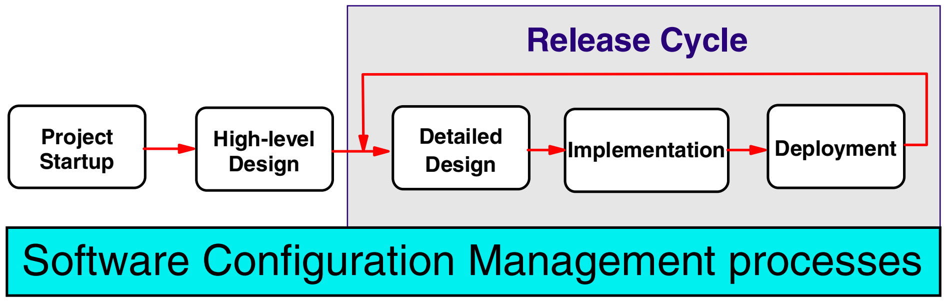


Figure 226. SCM and development process overview

This means that all work products that are created and or updated during the development process should be managed by the SCM processes and the toolset used to implement the required SCM functionality.

There will be many pieces to manage; not only the developed code artifacts such as .jsp, .java, .class, .servlet, .html, and .gif files, but also the business, functional, and database models involved, test cases, and runtime artifacts, such as .jar files, DLLs, and executables.

Moreover, versions of these artifacts do have relationships. A specific version of an application model relates to a specific version of the implementing code, test cases and test beds.

These related configurations of artifacts are usually referred to as being baselines or drivers (we will use the term baselines). Typically, you would have developer baselines, staging or development baselines (test, integration test, preproduction test), and production baselines.

While developers are focused on crafting the work products, they typically do not bother much about how to manage these baselines. The pressures of delivering work faster and faster, with higher quality, paired with the increasing complexity of the development environment, should give some food for thought in this respect.

Developer roles

Developers have different roles within the development process, and they have a different perspective, thus they have different requirements for SCM functionality.

In Figure 227 we illustrate that, in our opinion, SCM functions call for at least one additional development role.

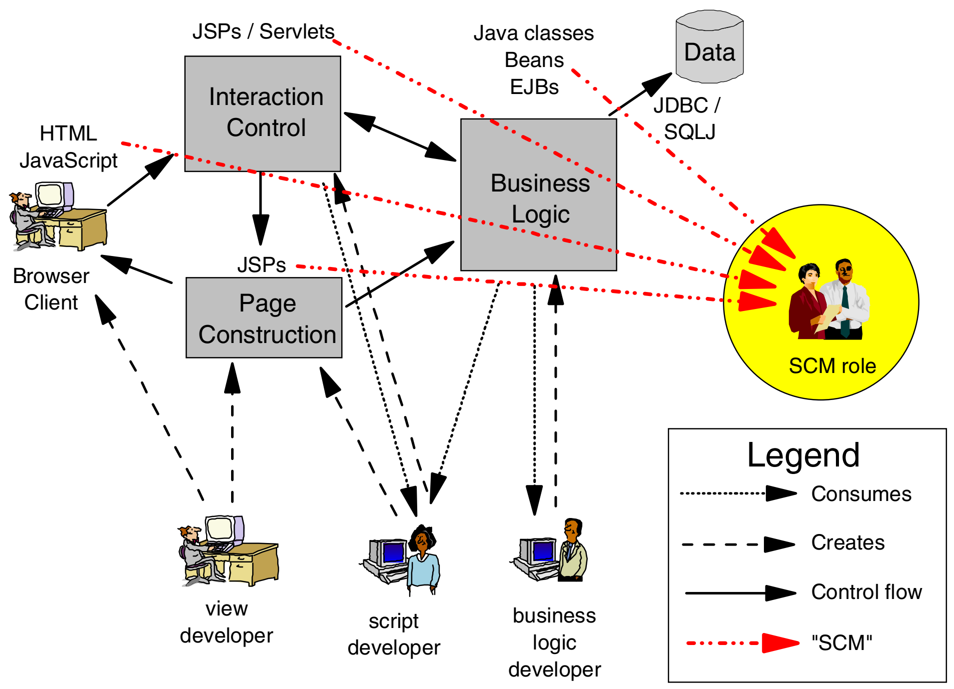


Figure 227. Application topology 1: additional SCM role

We introduce an SCM role to emphasize the importance of SCM for a development project. The person in this role should make sure that the developers are provided with SCM functionality, and should accept responsibility for making sure that the development baselines are managed throughout the development life cycle.

From this perspective, the SCM role has SCM requirements beyond those of the other developer roles. While the developers' SCM requirements perhaps could be fulfilled with a relatively simple tool or process (for example, copy before update), the requirements of the SCM role cannot be met easily. This provides one more reason to identify this role. This does not mean we should forget about the developers, they simply have other SCM requirements.

The SCM role is often embedded within the project leader's role, although given the complexity, we would advise assigning a separate developer for this role. Note that Figure 227 only covers the development perspective. From a runtime and production perspective, there will be more roles that have interest in SCM functionality.

Some aspects of the latter perspective are covered in the redbook *Managing Your Java Software with IBM SecureWay On-Demand Server Release 2.0*, SG24-5846.

We will focus on the development process SCM requirements, and thus focus on the development roles.

In an application topology with enterprise data, legacy systems, and third party applications, the SCM complexity is growing. There are more artifacts to manage, and artifacts may reside not only in the distributed environment, but also on a mainframe. This also means that build, deployment, and testing is more challenging (Figure 228).

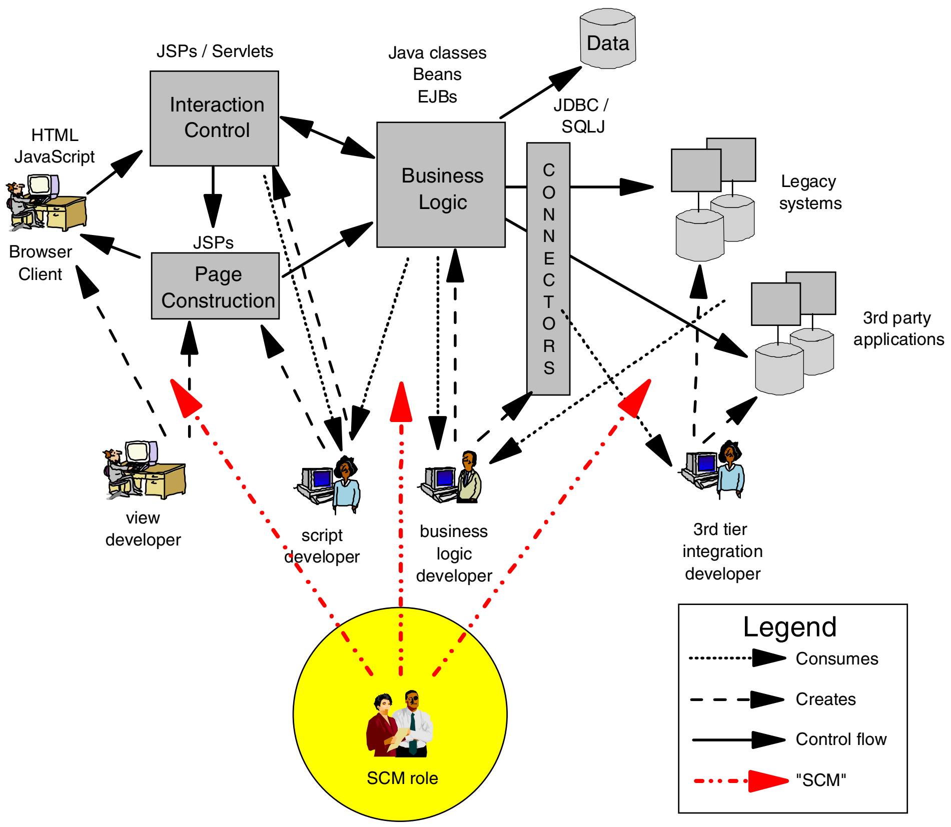


Figure 228. Application topology 2: more complexity

On top of that, it is very likely that multiple SCM tools have to inter-operate. Most organizations have mastered SCM from a mainframe perspective. They have deployed tools, and have implemented SCM processes in that area.

In order to be successful with topology 2 based development efforts, it will be key to integrate at least the SCM processes on both sides. It might well be that the organization will continue to use multiple tools to implement E2E SCM. Even if there were to be a single E2E tool set available, migration costs might prevent the implementation.

However, if possible, you should have a strategic objective to implement an integrated E2E SCM solution, not only from a process perspective, but also from a tools perspective.

For this book, we had to make a choice. Therefore, we will focus on the distributed environment, where a lot of organizations are struggling to get SCM in place.

It is not just a matter of defining processes and buying supporting tools; SCM should become a fact of development life. New technology that is thrown at development organizations comes with an associated new generation of developers, who will not necessarily appreciate the requirement for SCM.

Therefore, the SCM tools should at least be easy to use, and if possible, provide transparent SCM functionality from the developers' points of view, that is, from their IDEs.

The next section provides an example of SCM in the context of the WebSphere development work done for this redbook.

Implementing SCM aspects in a WebSphere Studio environment

In this section, we introduce various SCM aspects within our WebSphere environment. The result is that our sample code is version controlled by an external SCM tool in a manner that supports the SCM requirement to have a repeatable development process.

SCM aspects

The most important aspects of SCM are versioning and single point of control.

Versioning

Versioning of related artifacts is a foundation for all other SCM functions. In fact, this is a function developers understand. Before applying changes, you make a copy to be able to travel back in time to a situation when you knew things were working. If we can provide an easy-to-use function that supports versioning, that is an easy sell to a developer.

As stated before, other SCM stakeholders will have functional requirements beyond that. Project leaders or developers who have SCM roles assembling work from various developers to create a new version of the system or application will have different requirements.

In fact, depending on the role, developers will have other requirements as well:

- ❑Testers would like to be able to raise test environments, encompassing the right level of all required artifacts, be it code, test beds, or test cases.
- ❑Analysts would like to have impact analysis tools spanning all technology at hand, operating on various versions of the application or system.

In this chapter we will focus on providing a sound versioning capability for all roles involved, without forgetting the requirements other roles are having. We strongly suggest that you assign a separate developer to the identified SCM role. This developer must understand the SCM requirements of the complete development process to design one SCM solution to meet those requirements.

Single point of control

Within our development environment we are using both VisualAge for Java and WebSphere Studio. From a versioning perspective, only one of both tools can be the master.

Because WebSphere Studio provides an open environment, integrates all possible tools, and provides two-way integration with VisualAge for Java, we decided to use WebSphere Studio as the single point of control. Figure 229 shows that this is an obvious choice.

However, this means that the people performing the different roles identified should have a level of discipline as far as the process to follow is concerned. This is especially the case when you have to change existing Java code or craft new Java code.

The other development artifacts, given our choice, are by default controlled through WebSphere Studio.

The development of such procedures is required because multiple roles are stakeholders in the Java code, for example, view developers, script developers, and business logic developers (see Figure 228 on page 301).

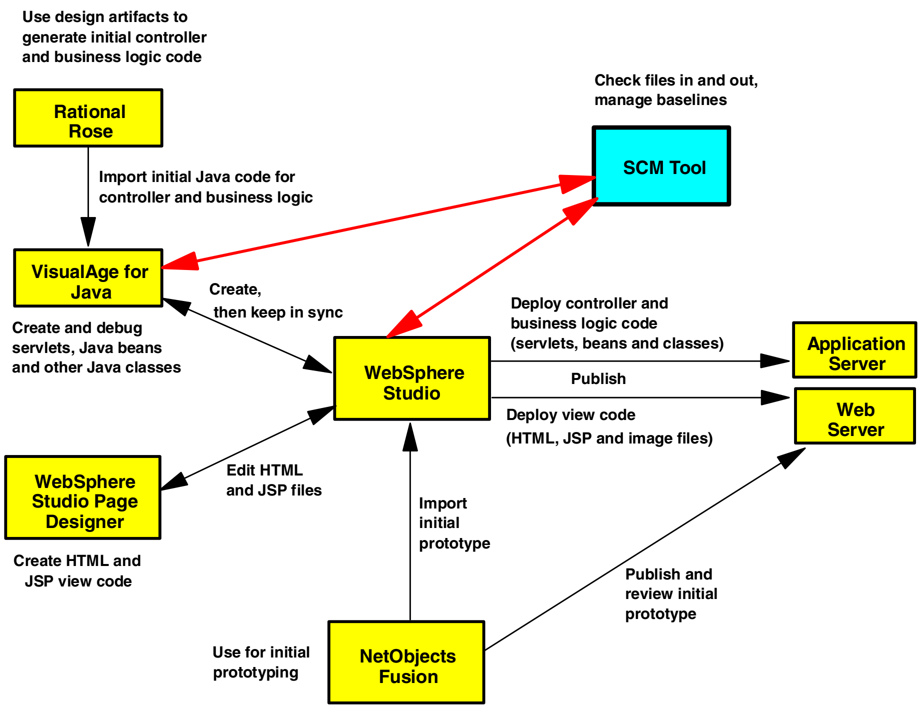

Use design artifacts to generate initial controller and business logic code
Rational Rose
Check files in and out, manage baselines
SCM Tool
Import initial Java code for controller and business logic
VisualAge for Java
Create, then keep in sync
Create and debug servlets, Java beans and other Java classes
Deploy controller and business logic code (servlets, beans and classes)
WebSphere Studio
Application Server
Publish
Deploy view code (HTML, JSP and image files)
Web Server
WebSphere Studio Page Designer
Edit HTML and JSP files
Import initial prototype
Create HTML and JSP view code
Publish and review initial prototype
Use for initial prototyping
NetObjects Fusion

Figure 229. Tools usage in the source code implementation phase

The requirement to keep the Java code artifacts synchronized between VisualAge for Java and WebSphere Studio is also illustrated in Figure 229.

To make things a bit more challenging, we have to consider as well that VisualAge for Java does have a team-based version management environment of its own, and because of the characteristics of developing fine grained object-oriented code, it should have those capabilities.

The team environment capabilities are covered in *VisualAge for Java Enterprise Version 2 Team Support*, SG24-5245.

The current level of the Software Configuration Control (SCC) API implementation in between VisualAge for Java with external SCM is not sufficient for our requirements. When using this integration, there is no indication whatsoever in the IDE that definitions are controlled by external SCM.

This implies that the synchronization effort must be manual, preferably performed by one role.

We would recommend that, given the Java task at hand, it is the responsibility of the project leader role or SCM role to load and create an open edition of the code within the VisualAge for Java repository. If a load of Java code is required, use the point-to-point integration capability of WebSphere Studio and VisualAge for Java.

> *You could argue that the start is an in-sync situation. In that case, you only have to create an open edition. We suggest that prior to that, you should run "compare" on the versions held in VisualAge for Java and WebSphere Studio.*

After exploiting the VisualAge for Java environment to create, change, and test the code (including the versioning of code increments during the development of the code) this should result in an edition that is ready to integrate with other artifacts being part of the same development effort.

It will be once again the responsibility of the project leader to move that edition of the code to the WebSphere Studio environment.

In the case of a servlet, this code might have to be integrated with a possibly changed JSP and deployed to the test environment, after which another iteration might be needed.

When everyone involved agrees, the project leader should create a version and move that version to the WebSphere Studio environment, after which the project can move to the next development stage in the development cycle.

For the time being, this synchronization effort is the pain you must endure if you would like to exploit the best of both worlds with a single point of control in mind.

In section "Working from WebSphere Studio" on page 325, we show that ClearCase has the capability to group related changes through its activities concept. This can be exploited to synchronize the Java code versions with JSP versions, thus providing additional support in the synchronization effort.

Note that this is true for the tactical time frame. Given IBM's SCM direction, the requirement for synchronization would still exist, but would be available when tools at hand are integrated with the SCM solution. Therefore, we suggest that you should not invest too much in creating automatic synchronization procedures, and instead use a procedural approach.

Note that the choice for project leader or SCM role to perform the synchronization task is arbitrary. It could just as well be the developer performing this task, depending on project and organization.

Choice for Clearcase as physical single point of control

We chose ClearCase as our SCM tool because it is positioned as the preferred tool for SCM. Its functionality will play an important role in the toolset implementing IBM's application for the e-business framework. Rational is a business partner signed up to integrate its toolset to this framework.

The relationship regarding SCM is even more fundamental. IBM will port ClearCase and ClearQuest to UNIX System Services (USS) on OS/390, thus creating a multiplatform toolset which can fulfill E2E SCM requirements.

Besides that, IBM will integrate the SCM functionality in future versions of the framework supporting development IDEs.

Selected functions of IBM's existing SCM offering will also be integrated within these SCM offerings.

However, this is not the only reason. ClearCase is also chosen because it does provide support to enforce an SCM oriented development process.

Rational SCM toolset

The SCM toolset from Rational includes ClearCase, ClearQuest, and Unified Change Management (UCM).

ClearCase

Rational ClearCase is a configuration management system designed to help software development teams track the files and directories used to create software. ClearCase enables you to manage the development and build process, and to enforce your site-specific development policies.

ClearCase is specifically designed to support parallel development, whether you are simply isolating the work of one developer from others on a small team, developing multiple releases in parallel using different teams, or sharing a source code base between multiple teams at geographically distributed sites.

ClearCase enables you to recreate the source base from which a software system was built, allowing it to be rebuilt, debugged, and updated, all without interfering with other development work.

In ClearCase, files and directories, or elements, are stored in a repository called a *versioned object base* or VOB. A version is a particular revision of a file or directory element.

Similar to many configuration management systems, ClearCase uses a “check-out, edit, check-in” model to manage software changes. When you check-out a file, ClearCase creates an editable copy, or checked-out version, in your view. When you check-in a file, ClearCase creates a new, permanent version of the file in the VOB.

You access and change elements using a view. A VOB contains all versions of a particular set of elements; a view selects a specific version of each element using a set of rules called a configuration specification (or config spec). The result is that when accessed through a view, a VOB looks just like an ordinary file system directory tree.

ClearQuest

Rational ClearQuest is a change request management application that allows you to track change requests for your products. Using ClearQuest, you can submit change requests, view and modify existing change requests, and create and run user-specific or site-specific queries and reports to determine the current state of your project.

In ClearQuest, change requests are stored as records in a ClearQuest relational database. Each record consists of all the data related to that record. ClearQuest supports different types of records for different projects and uses. For example, you might have record types for enhancements, defects, and activities, each with unique fields and data requirements.

ClearQuest records move through a pattern, or life cycle, from submission through resolution. In ClearQuest, each stage in this life cycle is called a state, and each movement from one state to another is called a state transition.

Unified Change Management

Rational Unified Change Management (UCM) combines ClearCase and ClearQuest to provide a complete, out-of-the-box, activity-based change management process.

UCM combines ClearCase configuration management capabilities (such as version control, parallel development, build management, and component-based management of directories and files) with ClearQuest change request and activity management capabilities (such as task management, state transition support, parent/child associations, policy enforcement rules, and extensive querying and reporting).

Our approach

Note that we will not write extensively on ClearCase concepts. We will only address ClearCase aspects briefly. Sometimes we will copy some descriptions.

The help information offered by ClearCase is both extensive and well structured. An approach that proves to be useful is to click on the *Help* button while performing the tasks we describe. Within these descriptions you will see hyperlinks to more information and, for instance, concept definitions that we do not want to replicate in this book.

Furthermore, ClearCase offers a fast path to a lot of information on Rational's ClearCase customers Web site. This site can be accessed by clicking on the *ClearCase on the Web* entry from the ClearCase administration console (Figure 230).

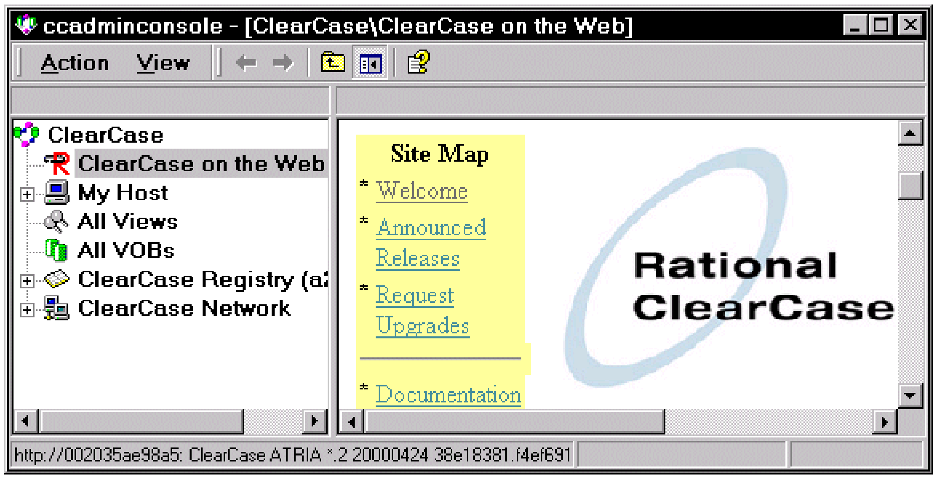


Figure 230. ClearCase on the Web from administration console

ClearCase in the WebSphere Studio environment

In this section we illustrate our approach of using ClearCase as the SCM tool in conjunction with WebSphere Studio and VisualAge for Java.

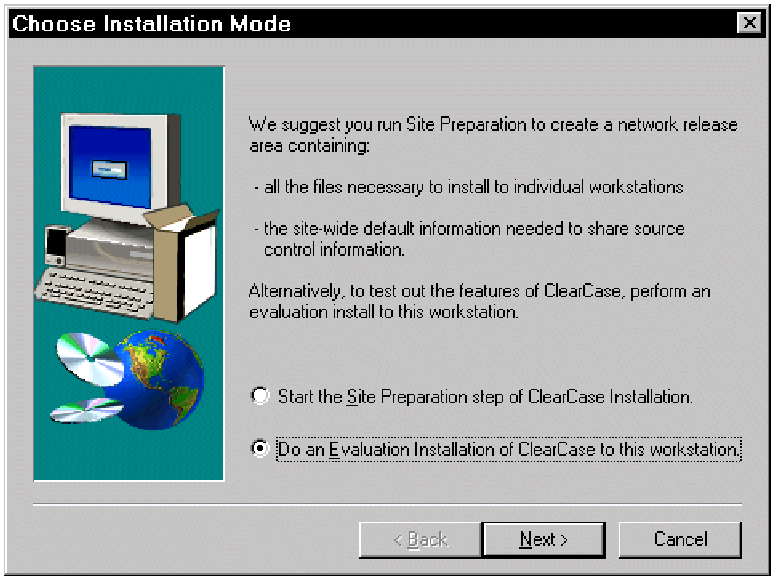
Installation

We installed ClearCase V4.0 in evaluation mode to avoid setting up a network installation of ClearCase.

A consequence of this approach is that you have to redo the installation, project setup, and project population steps when moving to a real installation. Therefore we suggest that you evaluate ClearCase by enabling one project (our sample).

Basically you have to execute the following steps to install ClearCase. We identified two starting points:

❑Use the autostart facility of the CD-ROM drive (Figure 231). Select the second radio button (evaluation install).



We suggest you run Site Preparation to create a network release area containing:

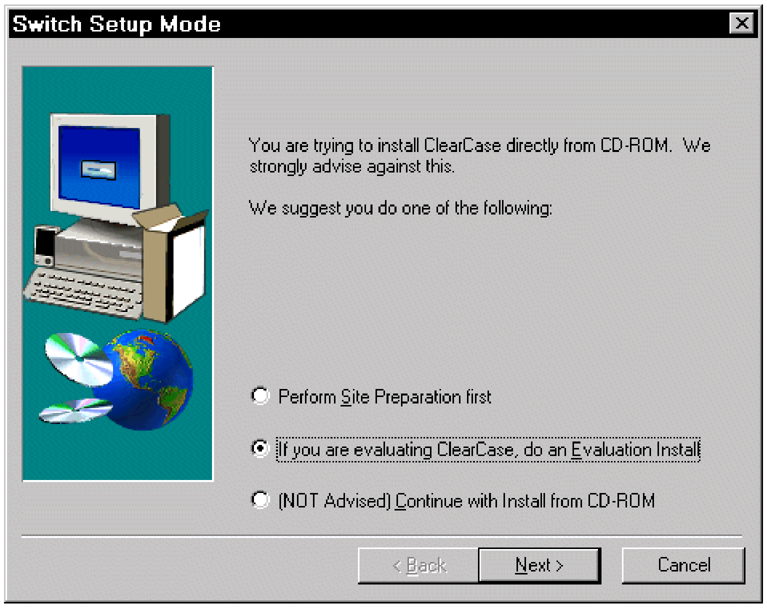
- all the files necessary to install to individual workstations
- the site-wide default information needed to share source control information.

Alternatively, to test out the features of ClearCase, perform an evaluation install to this workstation.

Start the Site Preparation step of ClearCase Installation. [✓ Do an Evaluation Installation of ClearCase to this workstation.]

Figure 231. ClearCase autostart installation mode panel

❑If for some reason the CD does not autostart, run the setup.exe from `cd_drive:\cpf\nt_i386` and the Switch Setup Mode dialog is displayed (Figure 232). Select the second radio button for an evaluation installation.


Switch Setup Mode

You are trying to install ClearCase directly from CD-ROM. We strongly advise against this.

We suggest you do one of the following:

( ) Perform Site Preparation first

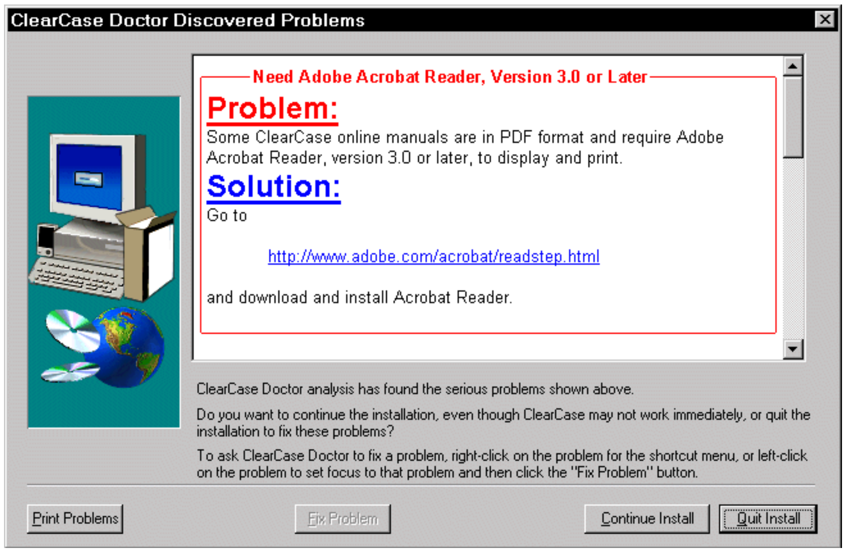
(●) If you are evaluating ClearCase, do an Evaluation Install

( ) (NOT Advised) Continue with Install from CD-ROM

Figure 232. ClearCase switch setup mode panel

❑On the Welcome to ClearCase install panel, click on *Next*.

❑Now the ClearCase Doctor screen is displayed (Figure 233).


ClearCase Doctor Discovered Problems

Need Adobe Acrobat Reader, Version 3.0 or Later

Problem:

Some ClearCase online manuals are in PDF format and require Adobe Acrobat Reader, version 3.0 or later, to display and print.

Solution:

Go to

http://www.adobe.com/acrobat/readstep.html

and download and install Acrobat Reader.

ClearCase Doctor analysis has found the serious problems shown above.

Do you want to continue the installation, even though ClearCase may not work immediately, or quit the installation to fix these problems?

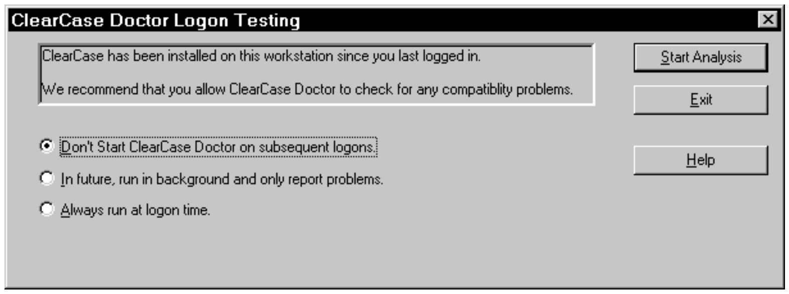
To ask ClearCase Doctor to fix a problem, right-click on the problem for the shortcut menu, or left-click on the problem to set focus to that problem and then click the "Fix Problem" button.

Figure 233. ClearCase Doctor Discovered Problems panel

You should look at the messages presented and possibly fix the problems after assessment. Note that if a TCP/IP DHCP problem is identified, it is not applicable for an evaluation installation. However, this might need to be considered when installing for real. We did assess all messages and decided that we did not have a real problem, and continued by clicking the *Continue Install* button.

❑After stepping through various panels, including the copying of files and reading the installation notes file, we clicked on *Finish* to reboot the machine.

❑After reboot, another Clearcase Doctor screen is displayed (Figure 234). We recommend selecting the first radio button to prevent starting of ClearCase Doctor at the next reboot.


ClearCase Doctor Logon Testing

ClearCase has been installed on this workstation since you last logged in.

We recommend that you allow ClearCase Doctor to check for any compatiblity problems.

[✓ Don't Start ClearCase Doctor on subsequent logons.]
In future, run in background and only report problems.
Always run at logon time.

Figure 234. ClearCase Doctor Logon Testing

❑Installation should be complete after this.

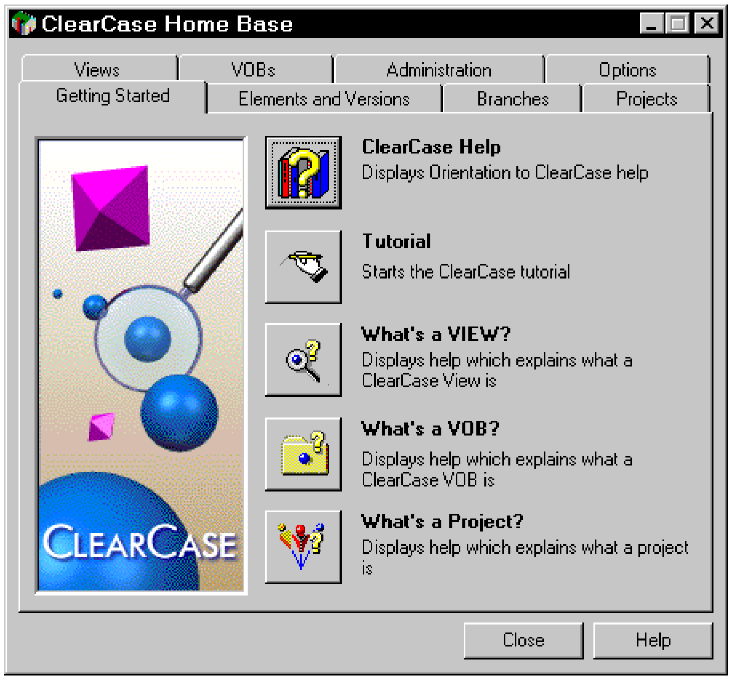
If you would like to de-install after the evaluation and prior to a real installation, you should run `cd_drive:\cpf\nt_i386\uninstal.exe`. Note that you must make sure by selecting the appropriate options that all information, including the variable directory, is removed. Not doing so results in confusing results after re-installation.

Testing the installation

After the installation is complete, there are various ways of working with ClearCase. Either go through the Windows start menu, ClearCase and ClearCase administration submenus, or double-click the ClearCase Home Base icon that should be on your desktop:

In the remainder of this chapter, we used the ClearCase Home Base route to complete the setup and configuration. After bringing a project under control of ClearCase, ClearCase will be used transparently from WebSphere Studio.

The ClearCase Home Base is shown in Figure 235.



ClearCase Help
Displays Orientation to ClearCase help

Tutorial
Starts the ClearCase tutorial

What's a VIEW?
Displays help which explains what a ClearCase View is

What's a VOB?
Displays help which explains what a ClearCase VOB is

What's a Project?
Displays help which explains what a project is

Figure 235. ClearCase Home Base

WebSphere Studio and ClearCase considerations

Because of the fact that we are in favor of following a structured process throughout the development life cycle, the obvious choice was made to exploit the process that is shipped with ClearCase. Furthermore, we kept our role approach in mind when making decisions on the implementation. Ease of use from a role perspective was also instrumental in making our implementation choices.

We will illustrate the integration using our redbook project. We suggest that you use the same project to evaluate ClearCase. We will take you through all the steps needed to enable the redbook project code as project within a ClearCase environment.

As indicated, ClearCase comes with an out-of-the-box process called *Unified Configuration Management* (UCM). This does not mean that this process cannot be changed, but everything is set up to support a development effort with a development process in mind. Throughout the setup steps documented in the following sections, we will comment on some aspects of UCM and the choices that we made.

Setting up a ClearCase project

To set up the ClearCase project environment, you have to create datastores containing project definitions and components.

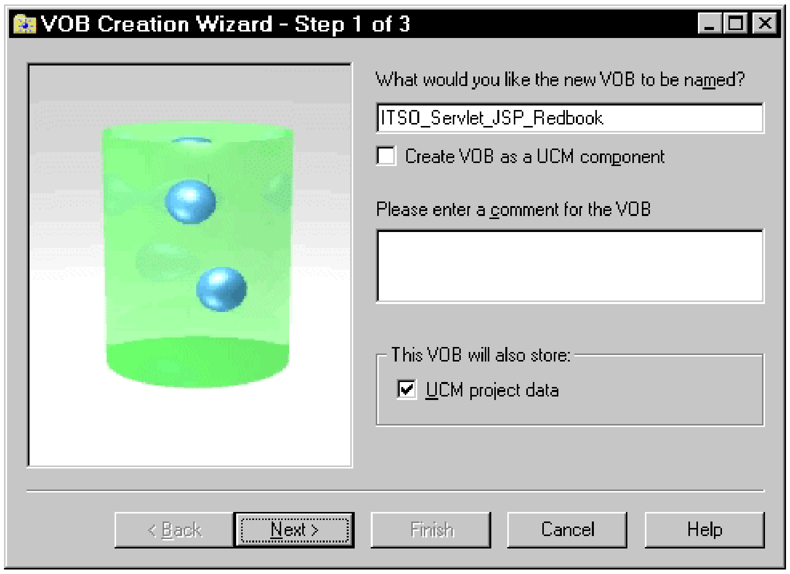
Creating a datastore: Project VOB (PVOB)

First of all ClearCase must have a place to store the project meta data. Recall that in ClearCase, the data stores are called versioned object bases (VOBs).

To create the PVOB, follow these steps:

❑In the Home base (Figure 235) select the VOB tab, and click the *Create VOB* button.

❑Enter the project name (*ITSO_Servlet_JSP_Redbook*) on the VOB Creation Wizard panel (Step 1 of 3), and make sure that only the *UCM project data* check box is selected (Figure 236). Click *Next*.


VOB Creation Wizard - Step 1 of 3

What would you like the new VOB to be named?
ITSO_Servlet_JSP_Redbook

☐ Create VOB as a UCM component

Please enter a comment for the VOB

This VOB will also store:
☑ UCM project data

Figure 236. ClearCase VOB Creation Wizard: project

Checking the *UCM project data* check box results in creating a VOB that only contains project meta data, and not actual development artifacts. This way, we separate interests. Probably the SCM role will be responsible for the content of this VOB (in consultation with the project leader and project management). The SCM role will tailor the project setup, including setup of component VOBs and developer views through which the developers can work with the component VOBs. We will discuss components and developer views when we create them in future steps.

❑On the next VOB Creation Wizard panel (Step 2 of 3), accept the defaults and click *Next*.

❑On the next VOB Creation Wizard panel (Step 3 of 3), click *Finish*.

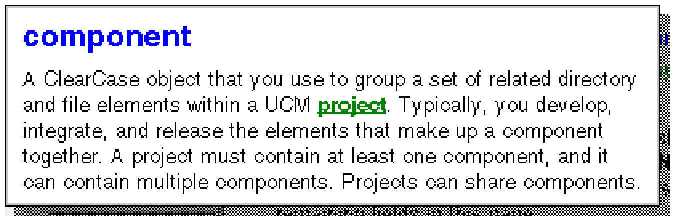
❑Then click *OK* in the confirmation panel, and after the processing is finished, you can click *Close* on the summary panel.

This completes the steps to create the project VOB.

Creating datastores: Component VOBs

After successful creation of the project VOB, you have to create one or more VOBs holding the project artifacts. We did choose an approach that separates our development artifacts in components.

Figure 237 shows the definition of a component.



component

A ClearCase object that you use to group a set of related directory and file elements within a UCM project. Typically, you develop, integrate, and release the elements that make up a component together. A project must contain at least one component, and it can contain multiple components. Projects can share components.

Figure 237. ClearCase Component definition

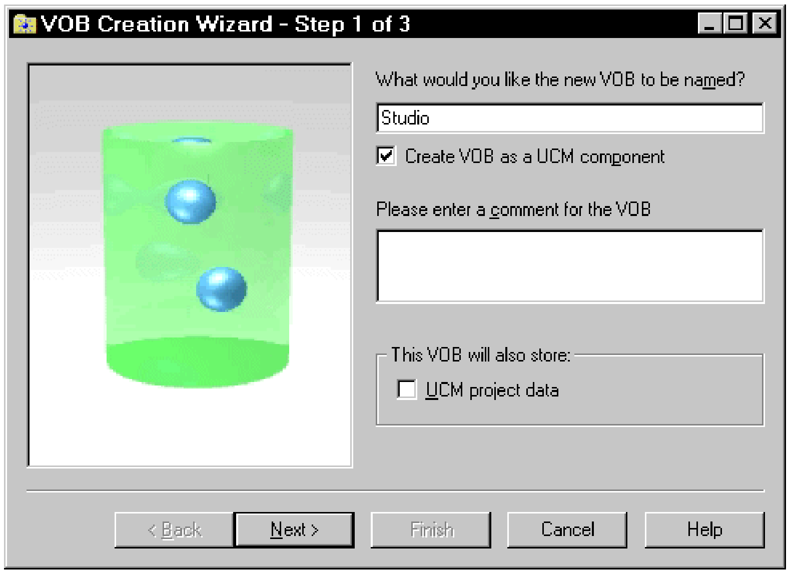
Actually, there are numerous valid approaches. One could store all artifacts in one VOB, or have multiple ones. We suggest that the choice of component VOBs should be guided by the roles identified and the tools that are used by these roles. The real separation of interests will be established with the view concept, which will be discussed later.

Knowing that we did choose WebSphere Studio as being the central point of control of all of our development artifacts, and having multiple roles with a requirement to have a broad working view on various artifacts, we made the decision to create a component VOB for the artifacts created and updated through the WebSphere Studio tool environment.

Definitely, this is an area you should plan for, and make proper decisions upon, after you have had more working experience. Note that the UCM approach is new in version 4.0 of ClearCase.

To create the component VOBs, you have to step through the same steps performed for the creation of the PVOB:

❑In the first step of the wizard (Step 1 of 3), select the *Create VOB as a UCM component* check box and enter *Studio* as the component name (Figure 238).


VOB Creation Wizard - Step 1 of 3
What would you like the new VOB to be named?
Please enter a comment for the VOB
This VOB will also store:
UCM project data
< Back
Next >
Finish
Cancel
Help

Studio

☑ Create VOB as a UCM component

☐ UCM project data

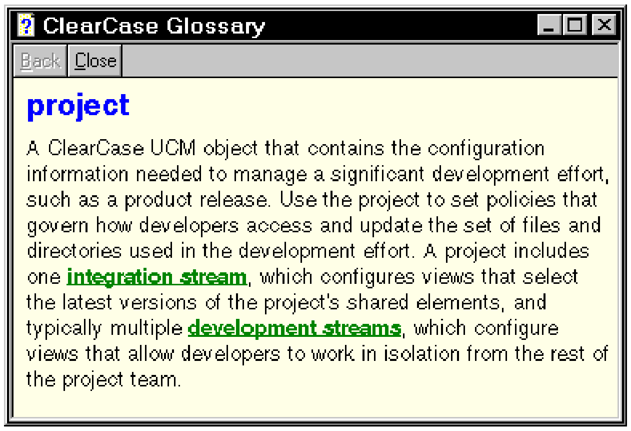
Figure 238. ClearCase VOB Creation Wizard: component

❑Run through the other steps in the same way as for the PVOB.

❑Repeat the process and create a component called *Rose*. We will describe the purpose of this component later.

Create the project

The next step is to create a ClearCase project. Figure 239 shows the definition of a project.


ClearCase Glossary

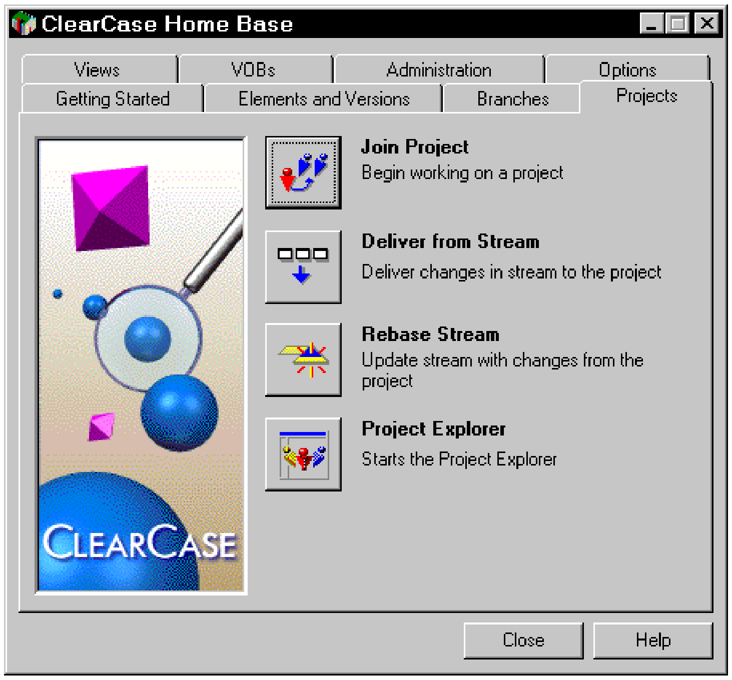
Back Close

project

A ClearCase UCM object that contains the configuration information needed to manage a significant development effort, such as a product release. Use the project to set policies that govern how developers access and update the set of files and directories used in the development effort. A project includes one integration stream, which configures views that select the latest versions of the project's shared elements, and typically multiple development streams, which configure views that allow developers to work in isolation from the rest of the project team.

Figure 239. ClearCase Glossary: project definition

❑Select the *project* tab in the ClearCase Home Base and click on the *Project Explorer* button (Figure 240).


ClearCase Home Base

Views | VOBs | Administration | Options

Getting Started | Elements and Versions | Branches | Projects

Join Project
Begin working on a project

Deliver from Stream
Deliver changes in stream to the project

Rebase Stream
Update stream with changes from the project

Project Explorer
Starts the Project Explorer

Figure 240. ClearCase Home Base: Projects

❑The Exploring ClearCase Projects window is displayed (Figure 241).

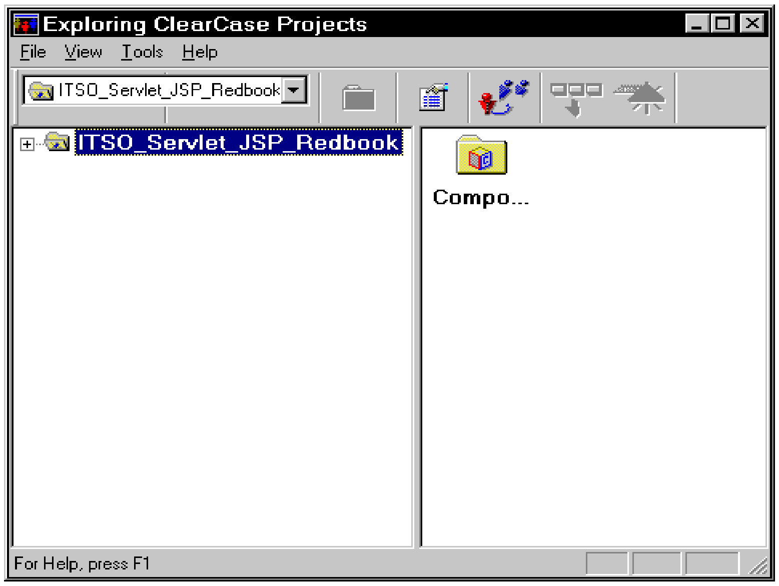
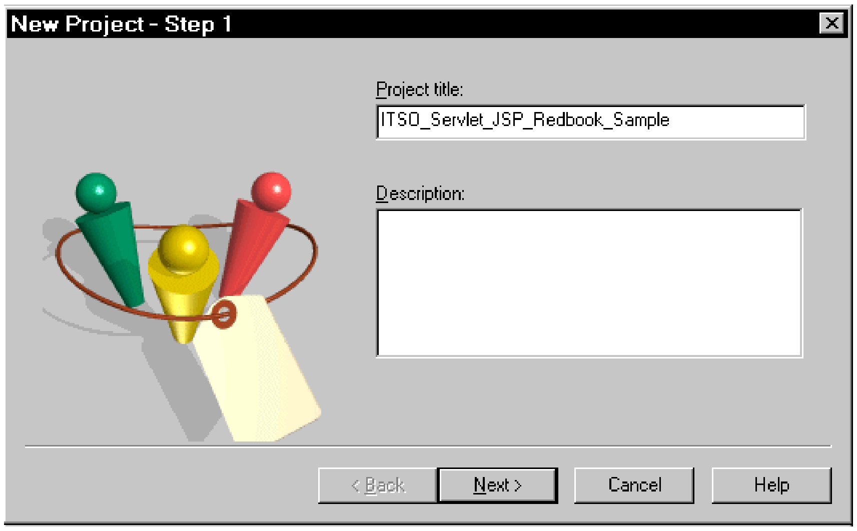


Figure 241. ClearCase project explorer

❑Select the ITSO_Servlet_JSP_Redbook VOB entry, then select *File -> New Project* and enter the project title (Figure 242). Click *Next*.


New Project - Step 1

Project title: ITSO_Servlet_JSP_Redbook_Sample

Description:

Figure 242. ClearCase create sample project

❑Leave the default (“no” radio button selected) and click *Next*. The Step 3 panel, where you can add component baselines, is displayed (Figure 243).

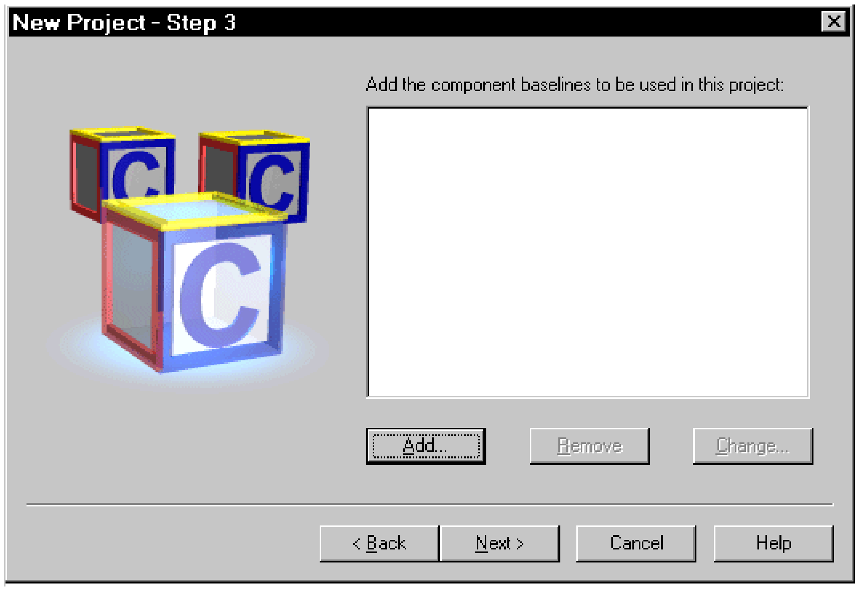
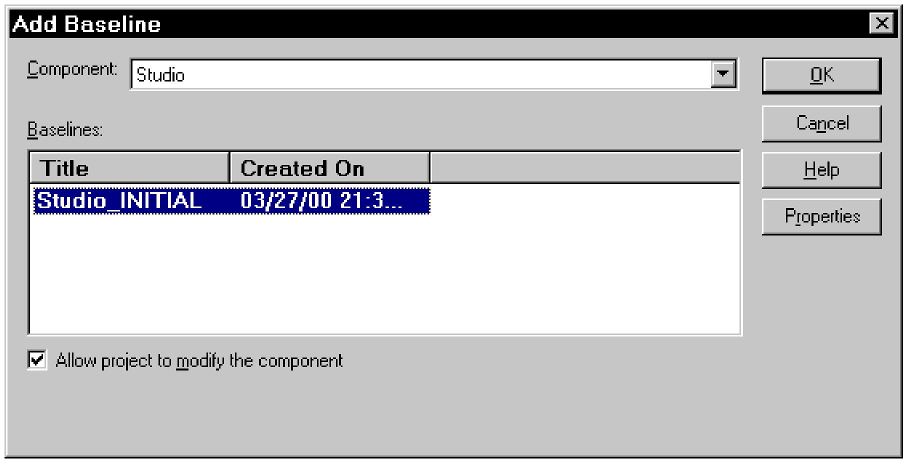


Figure 243. ClearCase create sample project (step 3)

❑Click *Add*, and the Add Baseline dialog is displayed. Select the *Studio_INITIAL* baseline (Figure 244).



Component: Studio

Baselines:

Title	Created On
Studio_INITIAL	03/27/00 21:3...

☑ Allow project to modify the component

Figure 244. ClearCase create sample project: add baseline

❑Click *OK*, and repeat the *Add* action for the *Rose* component.

Note that the actions performed in the previous steps provided you with an opportunity to choose a different starting point for your project. Although we only had an initial baseline, you could already have a production or tested baseline.

These standard baselines can be configured per installation and can be created by the project leader or SCM role.

Work from developers performed in private work areas (streams) can be moved to a shared work area (integration stream) that can be baselined at meaningful moments in time. This is a powerful concept to “stage” your development project.

These baselines can be the starting point for the developer’s work areas (views).

Also, note that rebasing is sometimes required to incorporate other people’s work or to incorporate versions of artifacts from baselines created after you baselined your work area. Thus, for example, you might incorporate a production version of a component in your view while you are still working with a tested version.

❑After adding the two components, click *Next* to get to the step 5 panel.

Note that this panel provides you with the opportunity to select if the activities of this project are to be managed by ClearQuest.

❑Click *Finish* to have the project defined. The result in the project explorer is shown in Figure 245.

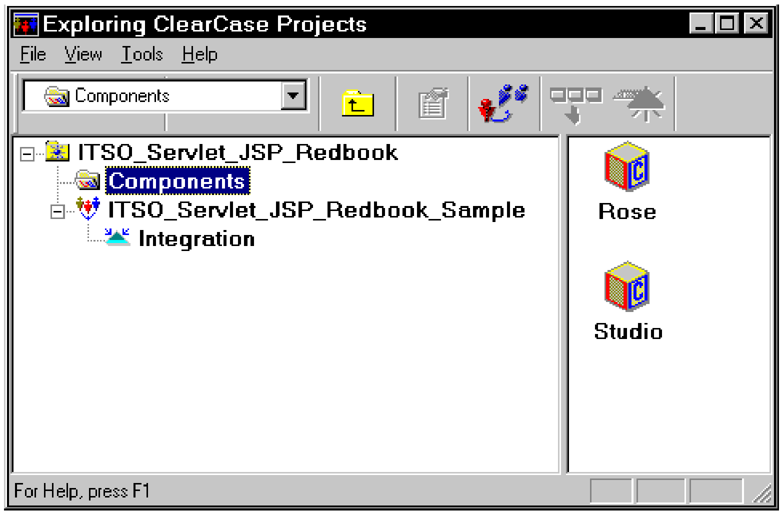


Figure 245. ClearCase project explorer after project creation

Create a view

The next step is to create a view. As indicated in “ClearCase” on page 306, a view is a ClearCase object that represents a work area for one or more developers.

By now you should be familiar with navigating in ClearCase. Furthermore, we will stick to the defaults all the way. So this task is only documented in text without screen captures. We recommend that you read the panels, though.

❑In the *Exploring ClearCase Projects* window, select the project, right mouse click, and choose *New -> Stream*.

❑In the Create a Development View panel, click *OK* to open a dialog for all the view options.

❑Step 1: The project is preselected; just click *Next*.

❑Step 2: Select *Reuse a Development Stream*, and click *Next*.

❑Step 3: Select *Create a Development View*, accept the proposed name, and click *Next*.

❑Step 4: Accept the proposed drive to connect to this view, and click *Finish*.

❑Click *OK* on the Confirm panel.

❑Click *OK* again to terminate the dialog.

The resulting project explorer window is shown in Figure 246.

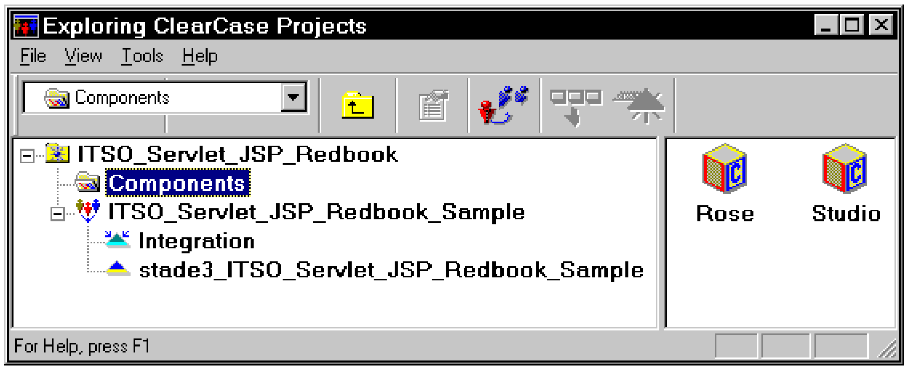


Figure 246. ClearCase project explorer project complete

Let’s discuss some aspects of what happened during the creation of the view.

Figure 247 shows the Windows explorer after the creation of the views.

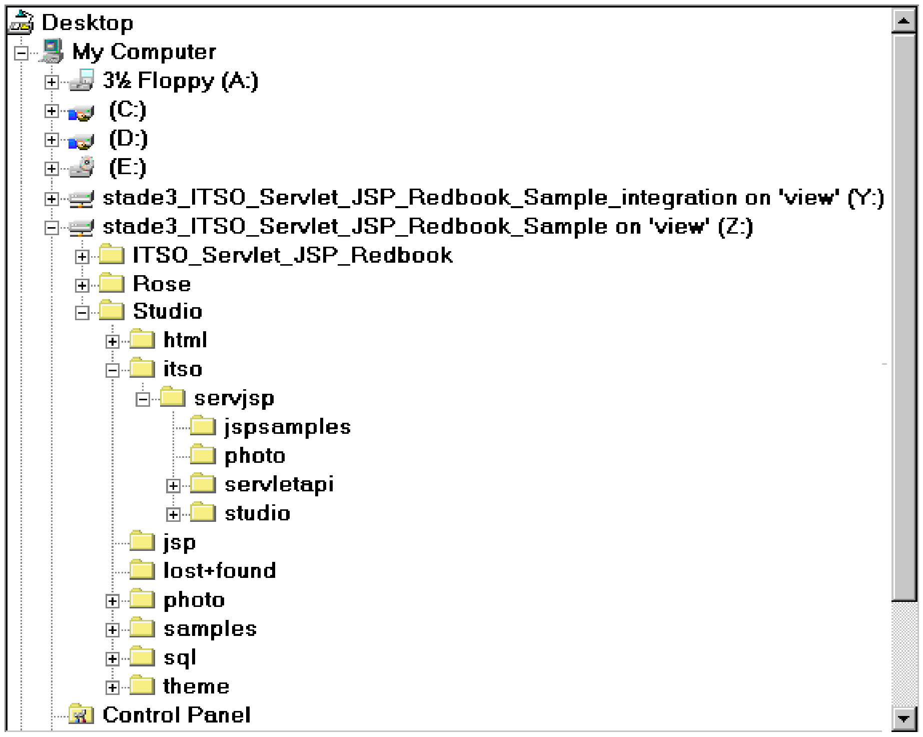


Figure 247. Windows Explorer view on views

Notice these views are linked as disks with letters starting backward from Z. The last view is expanded and you can see that there are three subfolders reflecting the project and the defined components.

These folders provide you with a dynamic view on the content of the real storage, that is, the VOBs in the ClearCase_Storage folder, which is located on your installation drive (for example, `C:\ClearCase_Storage\VOBs`).

In a normal distributed installation, these real VOBs will be placed on a secure ClearCase server somewhere in the network, and you would not see them. If you explore the content of these folders, you will see that the project folder contain information on the processes.

The other folders are empty for now. The next task at hand is to get development data into one of these folders, namely the Studio folder.

Enable ClearCase to the WebSphere Studio environment

Before you can populate the view, and in particular the Studio folder, you have to make sure that WebSphere Studio can operate through the views.

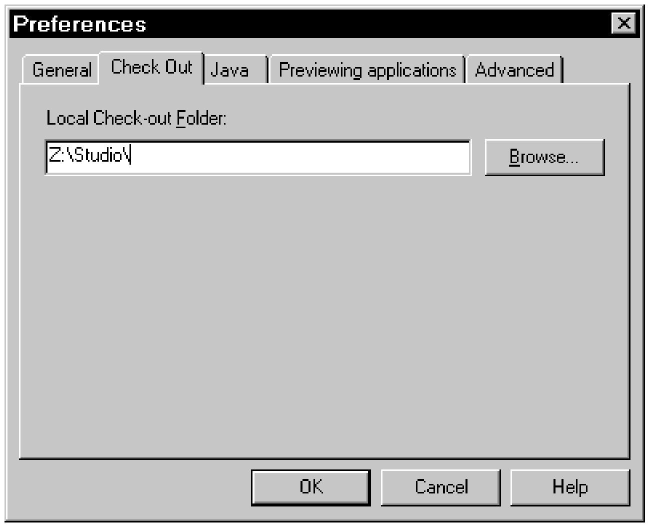
Note: Consulting the WebSphere Studio help files will not help you much, because the description there is quite cryptic! Instead, follow these instructions:

❑In WebSphere Studio select *Tools -> Preferences*.

❑Select the *Check Out* tab.

❑Fill-in the appropriate drive and component name (Figure 248).

If you have accepted the defaults when creating the views, it is likely that your view is connected on “Z” as well. If not, change this accordingly.


Preferences

Local Check-out Folder: Z:\Studio\

Figure 248. WebSphere Studio Tools Preferences: Check-Out

Now you are ready to let the SCM interface populate ClearCase with the project artifacts. Currently, the integration of WebSphere Studio with third party SCM tools is exploiting Microsoft’s Software Configuration Control (SCC) API.

The current thinking is that this proprietary API will be replaced by a standard committee endorsed standard such as Web-based Distributed Authoring and Versioning (WebDAV). For more information on WebDAV, see:

```
http://www.webdav.org                        <== WebDAV Resources
http://www.alphaworks.ibm.com/tech/DAV4J     <== WebSphere DAV for Java
```

Bring the projects artifacts under ClearCase control

To bring the project development artifacts under ClearCase control now takes only a few steps:

❑Open the project in WebSphere Studio first.

❑Select the project and *Project -> Version Control* (Figure 249). Note that WebSphere Studio has identified that ClearCase is present on the machine and presents you with a choice.

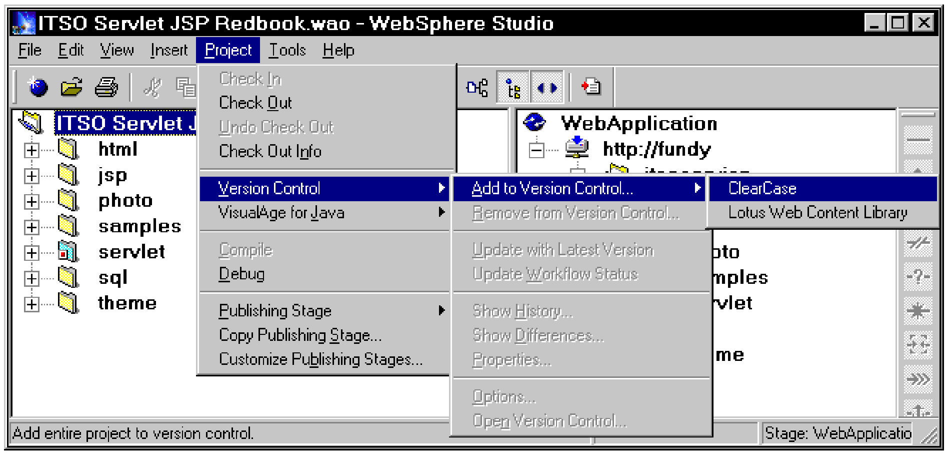
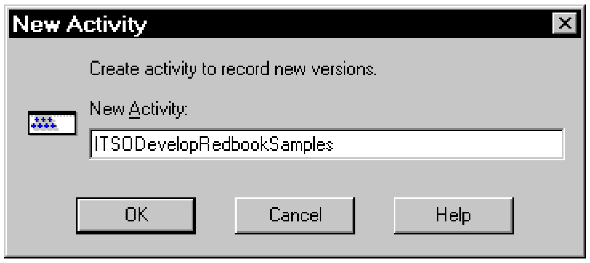


Figure 249. WebSphere Studio project version control

❑Select ClearCase, and processing begins. ClearCase prompts you to enter an activity. Typically this is the development task at hand; in our case the development of the Redbook samples (Figure 250).


New Activity

Create activity to record new versions.

New Activity:

ITSODevelopRedbookSamples

Figure 250. WebSphere Studio project version control activity prompt

❑Enter a new activity name and click *OK*. Processing takes quite a while to bring all the Studio components into the ClearCase repository.

This was pretty easy, wasn't it? And this is the way it should be!

What is an activity?

ClearCase groups and relates changes and associated versions to an activity. The project leader or SCM role can use these activities to move the changes associated with these activities and integrate them in the integration stream. Moreover, ClearQuest can be used to drive the activities that might need changes applied to various components through a defined process and track these throughout the development cycle.

In a normal life project, the project leader would now promote the defined artifacts to a production baseline, which could be the starting point for new development and maintenance. We are not changing anything, so we will stick to this initial baseline.

WebSphere Studio with external SCM

After processing has finished, you can tell from the WebSphere Studio window that the project artifacts are now controlled by external SCM, because black locks are displayed now behind the folders (Figure 251).

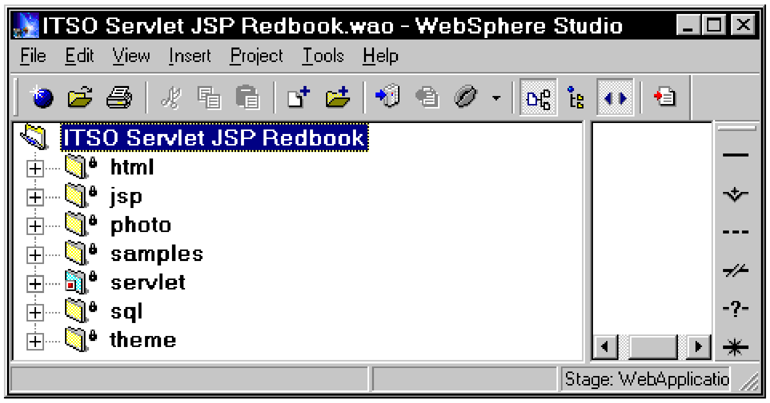

ITSO Servlet JSP Redbook.wao - WebSphere Studio
File Edit View Insert Project Tools Help
ITSO Servlet JSP Redbook
html
jsp
photo
samples
servlet
sql
theme
Stage: WebApplicatio

Figure 251. WebSphere Studio external version control GUI identification

In the next section, you will see that the use of SCM underlying WebSphere Studio is transparent (after set up, of course, which is one more reason to have a special SCM role who takes care of this without the developers needing to bother with it.)

Working from WebSphere Studio

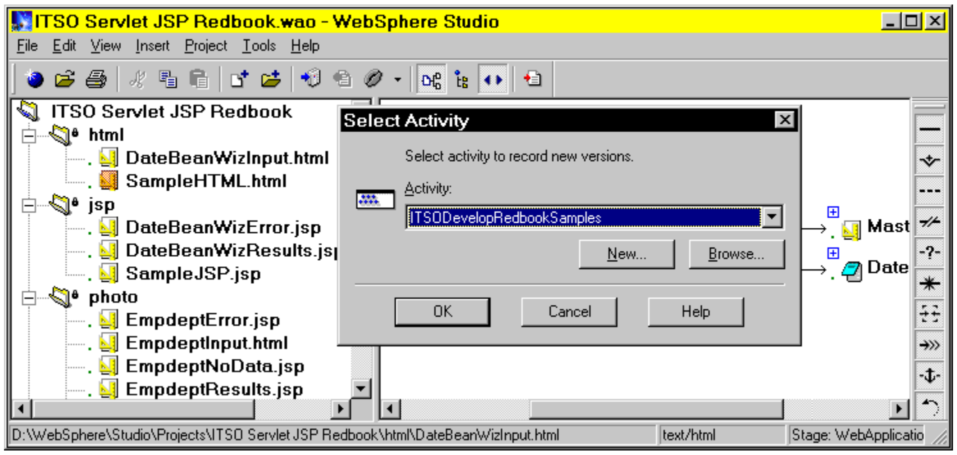
Working with WebSphere Studio means modifying artifacts with Studio tools. Each operation requires checking-out an artifact for modification, then checking-in this artifact when done.

Check-out

There is not much difference in the way developers would work with the development artifacts, as opposed to the situation when the check-outs are made to a shared disk. Refer to "Checking-out and checking-in files" on page 237 for more information.

The big difference, obviously, is that the external SCM is now taking care of locking and versioning.

Because of the fact that we have implemented ClearCase with UCM, the developers have to specify an activity when they want to work with a development artifact. This is illustrated in Figure 252. Installations that are set up without UCM do not prompt the user for this.


ITSO Servlet JSP Redbook.wao - WebSphere Studio

File Edit View Insert Project Tools Help

ITSO Servlet JSP Redbook
- html
  - DateBeanWizInput.html
  - SampleHTML.html
- jsp
  - DateBeanWizError.jsp
  - DateBeanWizResults.jsp
  - SampleJSP.jsp
- photo
  - EmpdeptError.jsp
  - EmpdeptInput.html
  - EmpdeptNoData.jsp
  - EmpdeptResults.jsp

Select Activity

Select activity to record new versions.

Activity: ITSODevelopRedbookSamples

New... Browse...

OK Cancel Help

Mast

Date

D:\WebSphere\Studio\Projects\ITSO Servlet JSP Redbook\html\DateBeanWizInput.html | text/html | Stage: WebApplicatio

Figure 252. WebSphere Studio project external version control check-out

Check-in

Another difference occurs when a developer wants to check-in an artifact the developer has been working on. At check-in time, a pop-up panel is presented. (Figure 253). The two options are obvious. We suggest that you select the second check box, because it does make sense to record why you are creating a new version.

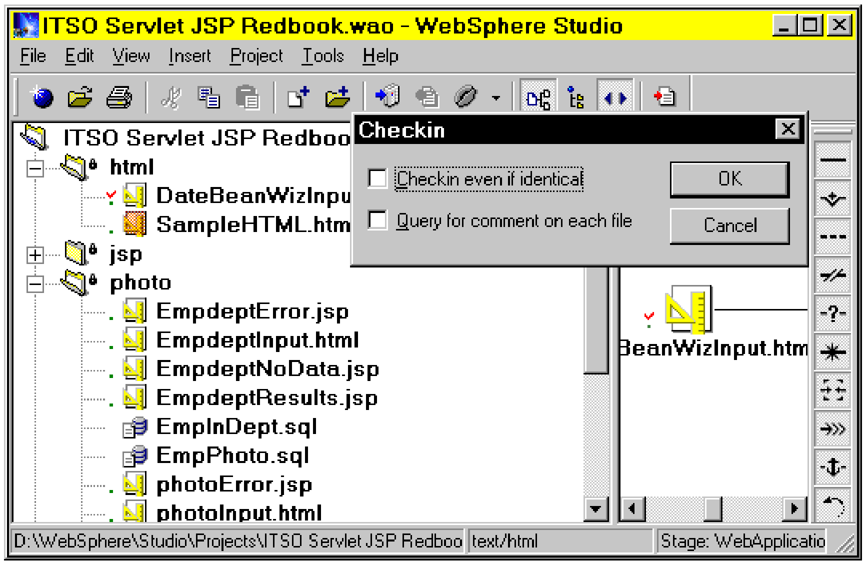


Figure 253. WebSphere Studio Project external version control check-in

Dependency relationships

If the artifact you want to operate on has dependencies, you will be asked if you want to preserve this relationship in the sense that the associated artifacts can be locked for the same activity as well.

Note that *undo check-out* is not implementing this behavior.

New artifacts and import from VisualAge for Java

If you define new artifacts, ClearCase prompts you to define the newly created artifact. This also means that on importing artifacts from VisualAge for Java, ClearCase prompts you to define the artifact.

Working with files directly

You can operate directly on the source files that are visible in the Studio component folders (from the Windows explorer). If you right-click on a file in these folders, you have more ClearCase functionality available to you than in the WebSphere Studio IDE. (This integration is limited by the capabilities of the SCC API.)

The pop-up menu listing the available functions is illustrated in Figure 254.

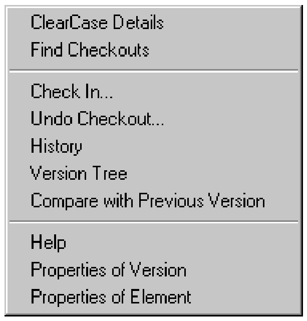


Figure 254. ClearCase direct functions from Windows Explorer

As far as we have explored, there is no limitation imposed on the capabilities you have within WebSphere Studio when working with ClearCase, as opposed to working on a project that is not controlled by ClearCase.

Note that implementations of other external SCM tools could well be different. It is up to the SCM tool vendor to decide which SCC capabilities to implement.

Reflections on SCM procedures

In “SCM aspects” on page 302, we indicated that you should have procedures in place to make sure that the developers operate on the correct version and deliver the right version.

Although we have provided some procedural guidelines for the interaction between VisualAge for Java and WebSphere Studio, this clearly is a moving target. These procedures will change as soon as you are implementing more and more SCM functionality. And, moreover, they are tool dependent as well.

This makes it difficult to write down explicit procedures for each case. If you would only deploy the ClearCase tool for versioning, without the project approach (UCM), the procedures would differ from a situation in which UCM is used, and it would differ from a situation when ClearQuest is implemented as well.

When other technologies are introduced and other associated development roles are identified, everything could change again.

We suggest that this calls for a pragmatic approach that avoids too much overhead. In particular, the developers want to produce artifacts instead of paperwork. The SCM role should make this happen by crafting a good E2E SCM design and implementing supporting tools.

WebSphere Studio and ClearCase in the broader SCM context

We started this chapter by defining and talking about aspects of SCM. In this last part of the chapter, we briefly comment on some of the other aspects in the context of the tools we used.

Build and deploy

In "Publishing stages and publishing targets" on page 247, we explain that WebSphere Studio has publishing capabilities.

At various moments within the development life cycle, it is required to deploy or redeploy the changed content of a Web site to the server where it should run. It is likely that this action needs to be synchronized with performing builds, creating baselines, and deploying (or publishing) actions.

We suggest that you exploit ClearCase to create a baseline, perform builds to create the baseline, and trigger the deployment/publishing operations. This can be done, because WebSphere Studio has APIs enabling you to trigger various publishing actions.

Activities and change management

If development teams are growing beyond just a few developers who can shout at each other, functions are required to track tasks and gather all changed artifacts together in project baselines.

This is even more true if, at the same time, the number of different roles and the number of associated tools were to increase. One way of accomplishing this is to use an integrated toolset providing those capabilities. ClearCase combined with ClearQuest, both of which play an integral role within the UCM approach, provide such integrated capabilities.

Rational Rose

Thus far we have not mentioned the *Rose* component that we created in addition to the Studio component.

We created this component to illustrate that it is easy to expand our sample implementation of ClearCase to other areas of the overall development effort and the associated roles. A systems analyst or designer using Rational Rose can exploit the ClearCase/ClearQuest combination for SCM functionality as well.

This way, we can expand the implementation to cover the whole proposed toolset for Topology 1 oriented development efforts (see Figure 229 on page 304).

Note that this is not limited to Rational and IBM development tools.

In fact, we know that there are more tools involved during the development process, such as documentation tools, discussion tools, and database tools. In Figure 255 we added some of these aspects to the Topology 1 tool diagram.

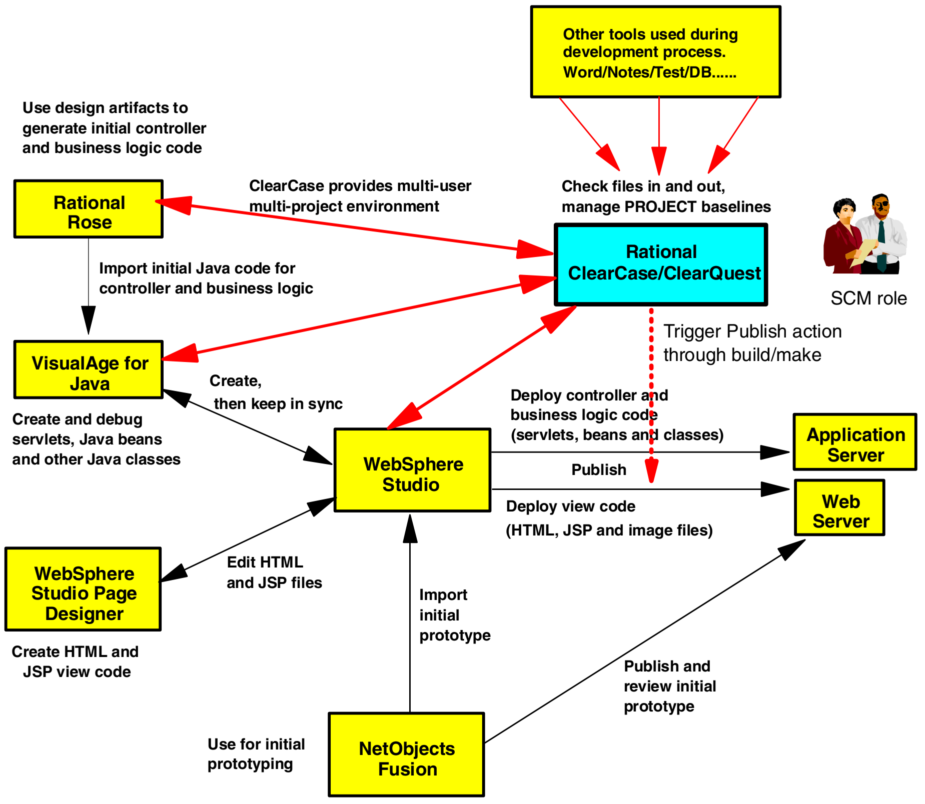

Other tools used during development process. Word/Notes/Test/DB......
Use design artifacts to generate initial controller and business logic code
Rational Rose
ClearCase provides multi-user multi-project environment
Check files in and out, manage PROJECT baselines
Rational ClearCase/ClearQuest
SCM role
Import initial Java code for controller and business logic
VisualAge for Java
Trigger Publish action through build/make
Create, then keep in sync
Create and debug servlets, Java beans and other Java classes
Deploy controller and business logic code (servlets, beans and classes)
WebSphere Studio
Application Server
Publish
Web Server
Deploy view code (HTML, JSP and image files)
WebSphere Studio Page Designer
Edit HTML and JSP files
Import initial prototype
Create HTML and JSP view code
Publish and review initial prototype
Use for initial prototyping
NetObjects Fusion

Figure 255. Topology 1 tools used during source code implementation

If the IBM/Rational alliance delivers an MVS (USS) version of ClearCase/ClearQuest, this toolset is well positioned to be able to manage Topology 2 development efforts from an E2E SCM perspective.

Epilog

If we look at Figure 226 on page 299, we indeed have barely scratched the surface in this chapter by covering only the implementation part of the process.

Our intention was to provide the reader with an example of how an important SCM aspect, version control, can be implemented with existing tools.

Looking at the environment created, we have the feeling that you will appreciate the functionality and integration provided. Furthermore, we argued that the implementation provided can be quickly expanded to cover more roles, more processes, and more SCM functions.

Besides this pragmatic objective, we conveyed the message that SCM is an important matter and should be handled accordingly (by SCM professionals having an SCM role in the development process.)

You really should target creating a firm E2E SCM implementation, building on top of the foundation we laid, to be successful in developing and deploying e-business applications at a pace needed by the business.

10 Web application design with servlets and JSPs

In this chapter we present a short overview of a guideline for designing Web applications consisting of servlets, JSPs, and JavaBeans.

Application structure

The general structure of a well-architected user interaction in a Web application is shown in Figure 256.

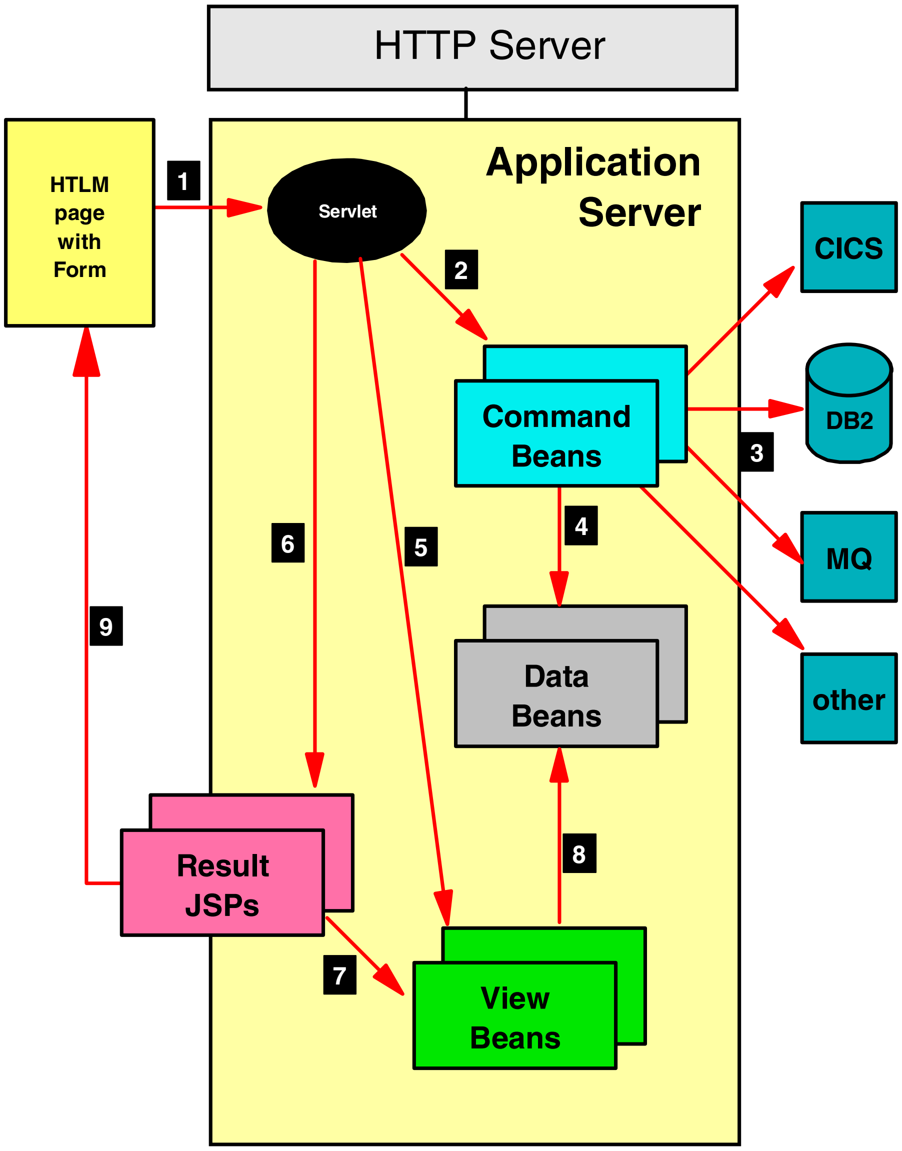


Figure 256. Web application design overview

The major parts of such a design are discussed in the sequence of the flow of the application.

HTML page

The input page for each step is either a static HTML page or a dynamic HTML page created from a previous step. The HTML page contains one or multiple forms that invoke a servlet for processing of the next interaction.

Input data can be validated through JavaScript in the HTML page or passed to the servlet for detailed validation.

Servlet

The servlet gets control from the Application Server to perform basic control of flow. The servlet validates all the data, and returns to the browser if data is incomplete or invalid.

For valid data, processing continues. The servlet sets up and calls command beans that perform the business logic.

The servlet initializes the view beans and registers them with the request block so that the JSPs can find the view beans.

Depending on the results of the command beans, the servlet calls a JSP for output processing and formatting.

Command beans

Command beans control the processing of the business logic. Business logic may be imbedded in the command bean, or the command bean delegates processing to back-end or enterprise systems, such as relational databases, transaction systems (CICS, MQSeries, IMS, and so forth).

A command bean may perform one specific function, or it may contain many methods, each for a specific task. Command beans may be called Task Wrappers in such a case.

Results of back-end processing are stored in data beans.

Data beans

Data beans hold the results of processing that was performed by the command bean or by back-end systems. For example, a data bean could contain an SQL result or the communication area of a CICS transaction.

Data beans may not provide the necessary methods for a JSP to access the data; that is where the view beans provide the function.

View beans

View beans provide the contract between the output producing JSPs and the data beans that contain the dynamic data to be displayed in the output.

Each view bean contains one or multiple data beans and provides tailored methods so that the JSP has access to the data stored in the data beans.

JSPs

The JSPs generate the output for the browser. In many cases that output again contains forms to enable the user to continue an interaction with the application.

JSPs use tags to declare the view beans. Through the view beans, the JSP gets access to all the dynamic data that must be displayed in the output.

Model-View-Controller

This design follows the Model-View-Controller design pattern:

- The JSPs (and HTML pages) provide the view.
- The servlet is the controller.
- The command beans represent the model.

The data beans contain the data of the model, and the view beans are helper classes to provide a data channel between the view and the model.

The servlet (controller) interacts with the model (the command beans) and the view (the JSPs). The servlet controls the application flow.

Detailed information

For detailed information about Web application design, refer to:

- The patterns for e-business described in Chapter 12, "Using Patterns for e-business to build the PDK" on page 347.
- The redbook: *Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

Part 2 Pattern Development Kit: a sample application

We will now walk you through a complete application. This application has been created to demonstrate a recommended design pattern.

We will take you through the purpose of the application, the design decisions involved, and then finally cover how to run it under both development and production environments.

Throughout this Part, the Pattern Development Kit will be referred to as the PDK.

11 Pattern Development Kit overview

In this chapter we will provide a brief overview of the Pattern Development Kit (PDK). We will then walk you through the application's front-end.

The underlying design of the application is discussed in Chapter 12, "Using Patterns for e-business to build the PDK" on page 347.

For detailed information about design patterns used in the PDK, refer to *Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

Background

The Patterns for e-business aim to communicate in a highly accessible fashion the business patterns, systems architecture (application and runtime topologies), product mappings, and guidelines required for different classes of applications. For the User-to-Business patterns there is also an associated Pattern Development Kit, which provides sample application code to illustrate effective use of those patterns.

For more information visit the **Patterns for e-business Web site** at `http://www.ibm.com/software/developer/web/patterns/`.

Application description

The PDK allows you to view some interplanetary weather data, although we should probably stress that the data is not completely authentic! The application behind the PDK is fairly straightforward, as it is merely a vehicle for demonstrating the patterns that have been used.

In the full version of the PDK, the weather data will be retrieved from a number of back-end systems and transports including DB2, MQ Series, CICS, and IMS.

However, for the purposes of this book, we are dealing with a subset of the application. We communicate with the SecureWay Directory for logon, but we do not connect to any of the subsystems. The infrastructure to connect to the other back-end systems exists in the code, but to keep this application simple, the systems themselves have not been included.

We modified the code to return dummy data to simulate IMS, CICS, and MQSeries.

Application walkthrough

To give you a better understanding of the application, we will first do a walkthrough using the functionality of the PDK. To be able to do this on your own machine, you will need to complete the instructions found in Chapter 13, “Running the PDK in WebSphere” on page 363.

Welcome page

The starting page of the PDK (Figure 257) is merely a front door to the application. From here you can click on the jigsaw piece graphic to go through to the PDK's home page.

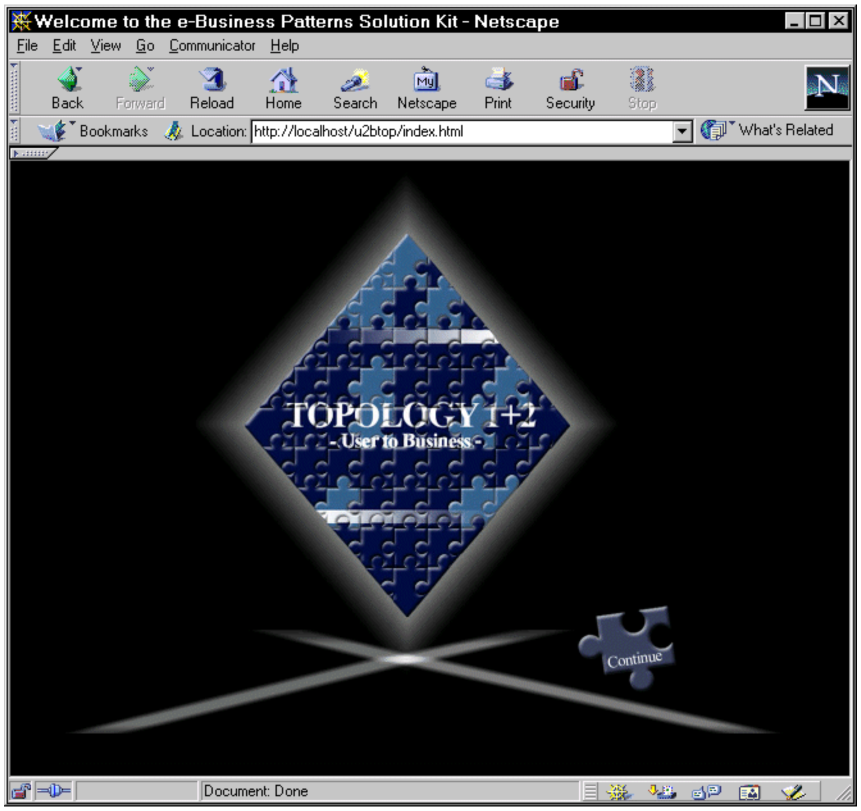

Welcome to the e-Business Patterns Solution Kit - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://localhost/u2btop/index.html
What's Related
TOPOLOGY 1+2
- User to Business -
Continue
Document: Done

Figure 257. Application welcome page

Home page

This page provides a very brief description of the PDK, including a quick overview of the two topologies being used. For more details on these topologies and the design patterns behind them, refer to Chapter 12, "Using Patterns for e-business to build the PDK" on page 347.

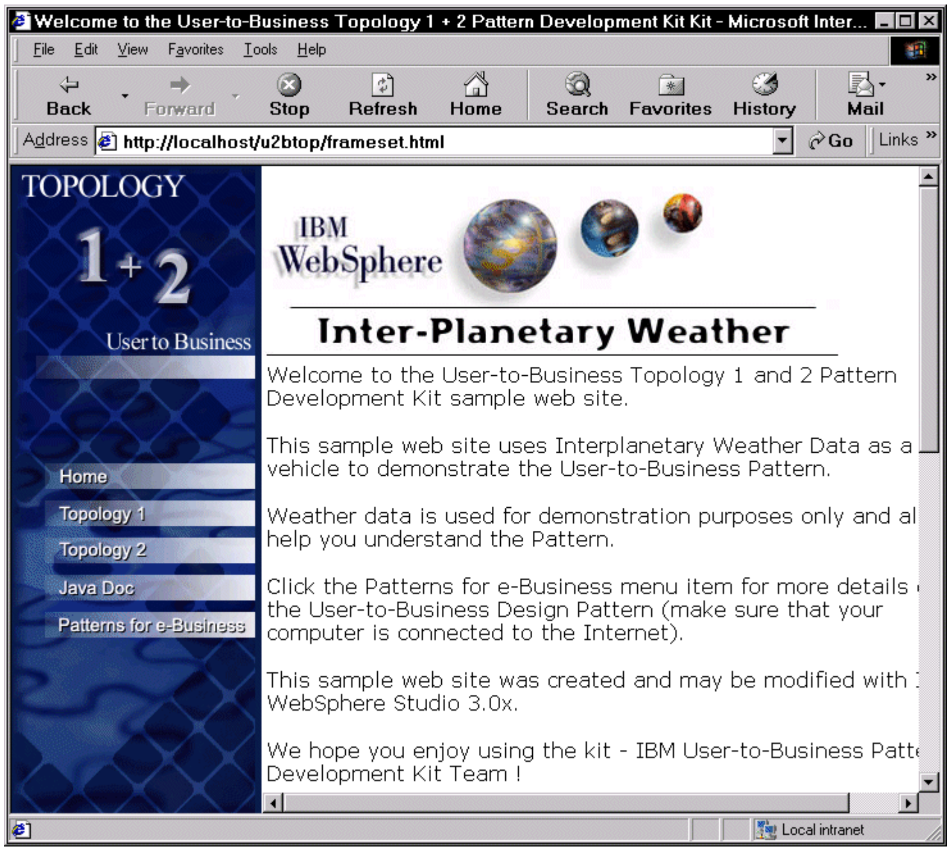
Available links

From this page (Figure 258) you can navigate to the two levels of application that are supported in this book:

❑Topology 1: Access to relational database on the Web server

❑Topology 2: Access to back-end enterprise systems (CICS, MQSeries, IMS)

If you choose one of these two links from the left-hand frame, then the application appears in the right-hand pane.


Welcome to the User-to-Business Topology 1 + 2 Pattern Development Kit Kit - Microsoft Inter...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://localhost/u2btop/frameset.html
Go Links
TOPOLOGY 1+2 User to Business
Home
Topology 1
Topology 2
Java Doc
Patterns for e-Business
IBM WebSphere
Local intranet

Inter-Planetary Weather

Welcome to the User-to-Business Topology 1 and 2 Pattern Development Kit sample web site.

This sample web site uses Interplanetary Weather Data as a vehicle to demonstrate the User-to-Business Pattern.

Weather data is used for demonstration purposes only and al help you understand the Pattern.

Click the Patterns for e-Business menu item for more details the User-to-Business Design Pattern (make sure that your computer is connected to the Internet).

This sample web site was created and may be modified with WebSphere Studio 3.0x.

We hope you enjoy using the kit - IBM User-to-Business Patt Development Kit Team !

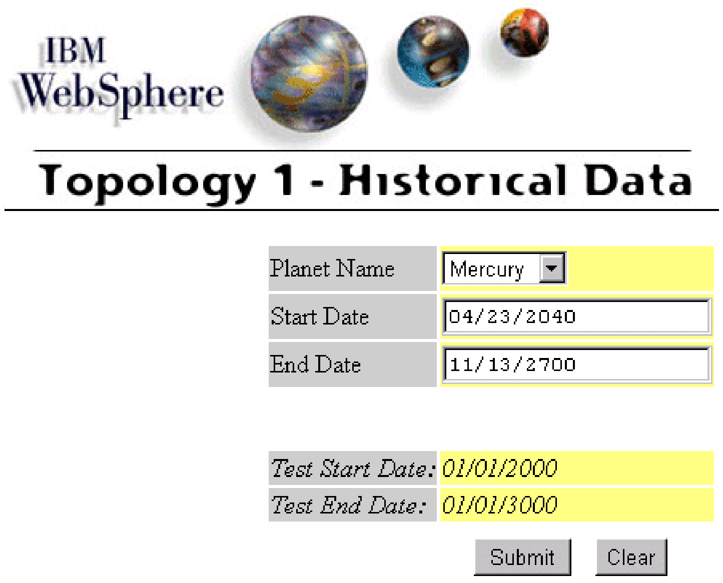
Figure 258. The application's home page

In the screen captures that follow, we only show the right frame. The left frame does not change during the application run.

Topology 1 — historical data

This application allows you to retrieve any historical weather data that has been stored about the planets. At the top of this page is the application itself, while beneath is a description of what is happening behind the scenes. For more details on this, see “Design techniques used” on page 353.

Select a planet, enter a valid date range and click on *Submit* (Figure 259). You should then be returned all the valid weather data on that planet for the given time period.



Planet Name	Mercury
Start Date	04/23/2040
End Date	11/13/2700
Test Start Date:	*01/01/2000*
Test End Date:	*01/01/3000*

Figure 259. Topology 1: input page

If you enter any invalid data into the fields, (including not completing a field), then an appropriate error message is returned. Also, if there is no weather data available for the date range that you provided, then a separate page will appear.

Returned data

If your request for data is successful, the application returns the data in one of two formats:

❑XML

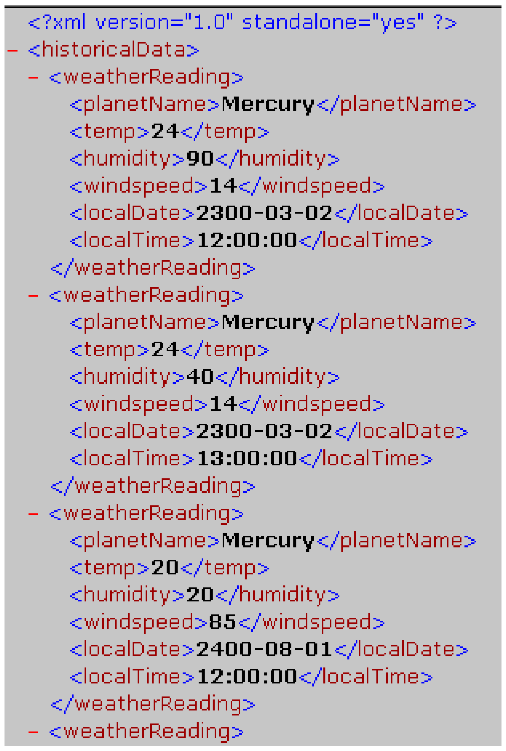
❑HTML table

XML

If your browser is XML enabled, then the weather data is returned in an XML format (Figure 260). Although this may not look particularly nice at this stage, it provides the browser with greater flexibility about how it may wish to display this information. You could have a designer create an appropriate XSL style sheet, and combine this with the output to create nicely formatted HTML.

For more information about XML and its uses, refer to the redbook "*The XML Files: Using XML and XSL with IBM WebSphere 3.0*", SG24-5479**.**

Internet Explorer 5.0 is currently the only mainstream browser that is XML-enabled.

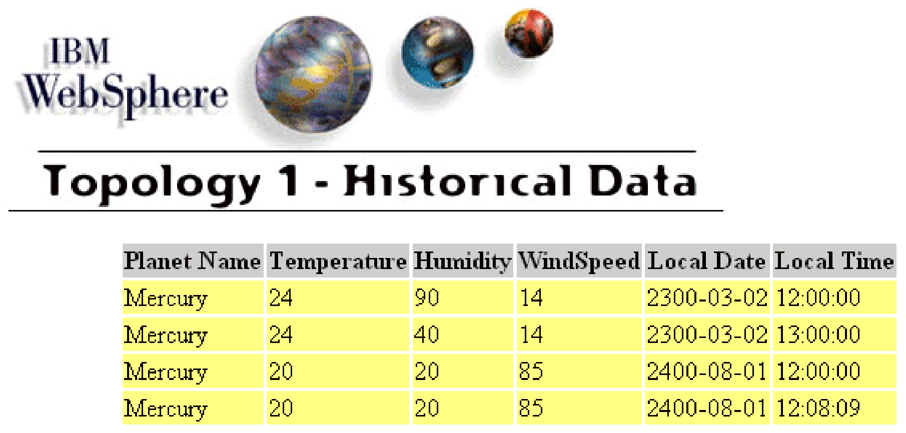


```
  <?xml version="1.0" standalone="yes" ?>
- <historicalData>
  - <weatherReading>
      <planetName>Mercury</planetName>
      <temp>24</temp>
      <humidity>90</humidity>
      <windspeed>14</windspeed>
      <localDate>2300-03-02</localDate>
      <localTime>12:00:00</localTime>
    </weatherReading>
  - <weatherReading>
      <planetName>Mercury</planetName>
      <temp>24</temp>
      <humidity>40</humidity>
      <windspeed>14</windspeed>
      <localDate>2300-03-02</localDate>
      <localTime>13:00:00</localTime>
    </weatherReading>
  - <weatherReading>
      <planetName>Mercury</planetName>
      <temp>20</temp>
      <humidity>20</humidity>
      <windspeed>85</windspeed>
      <localDate>2400-08-01</localDate>
      <localTime>12:00:00</localTime>
    </weatherReading>
  - <weatherReading>
```

Figure 260. Topology 1: output for XML enabled browsers

HTML table

For other browsers that are not XML enabled, the weather data is displayed as an HTML table (Figure 261).



Planet Name	Temperature	Humidity	WindSpeed	Local Date	Local Time
Mercury	24	90	14	2300-03-02	12:00:00
Mercury	24	40	14	2300-03-02	13:00:00
Mercury	20	20	85	2400-08-01	12:00:00
Mercury	20	20	85	2400-08-01	12:08:09

Figure 261. Topology 1: output for all other browsers

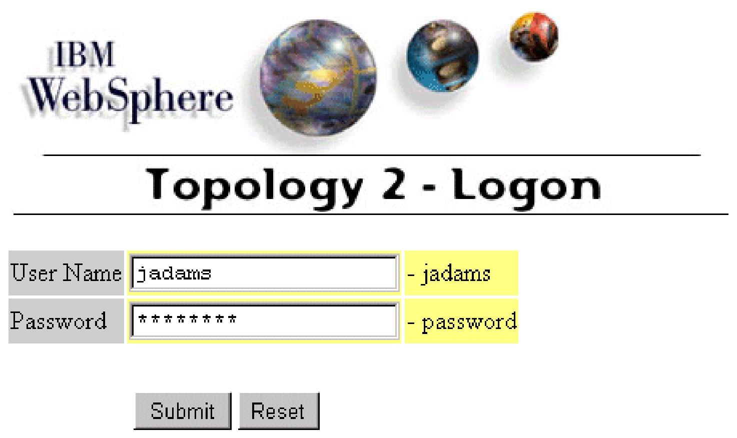
Topology 2 -— visit planets

This part of the PDK has security applied to it, so when you first follow the *Topology 2* link, you are asked if you accept the certificate that is being sent. As we trust the owners of this application (and its certificate), we choose to accept it. Depending on the browser, you have to go through a few warning dialogs before being allowed to accept that certificate.

Logon

When you are passed any certificate-related queries, you are prompted by the logon page (Figure 262). Here you need to enter the details of a user on the system.

To make things easier, a valid user ID (*jadams*) and password (*password*) combination are already listed on this page, to the right of the input fields. Enter these values in the fields and click on the *Submit* button.



User Name	jadams	- jadams
Password	********	- password

Figure 262. Topology 2: logon

Validated logon

The user ID and password are validated against the LDAP directory. Based on the type of user, the allowed menu options are retrieved from the database and displayed in the response (Figure 263).[1]

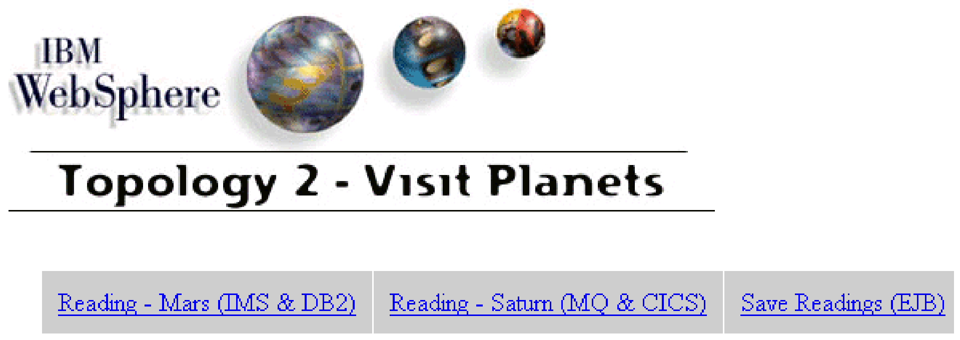

IBM WebSphere
Topology 2 - Visit Planets
Reading - Mars (IMS & DB2)
Reading - Saturn (MQ & CICS)
Save Readings (EJB)

Figure 263. Topology 2: weather readings options

From here, you can follow the three options:

❑Use an IMS connector to retrieve new weather readings from an IMS system.

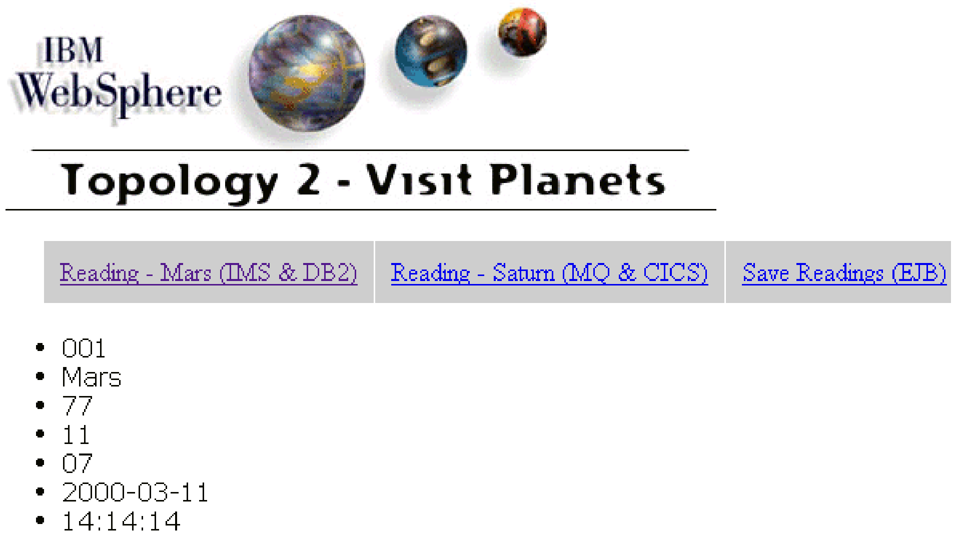
❑Use CICS and MQSeries connectors to retrieve new weather readings from CICS and MQSeries products.

❑Use an EJB to save the new weather readings in a DB2 table.

[1] If logon is not successful, make sure that the SecureWay Directory server has been started.

IMS connector

The real PDK application ships with an IMS simulator. For our purpose, we did not install the IMS simulator, and we modified the code to return dummy data (Figure 264).

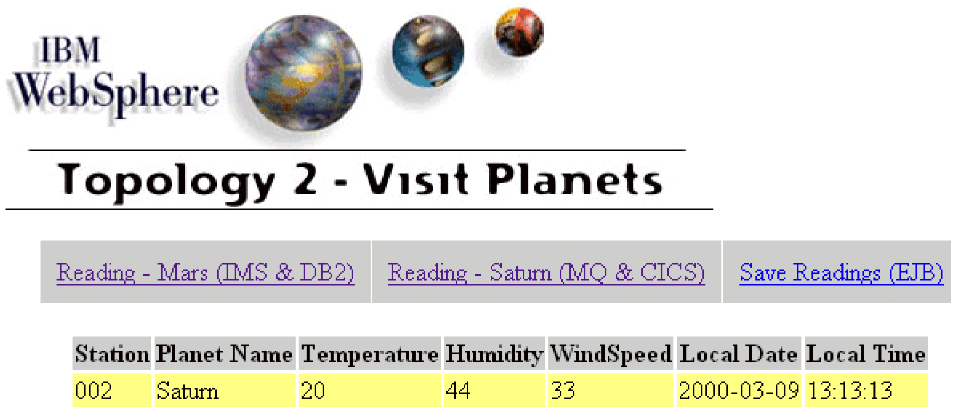

IBM WebSphere
Topology 2 - Visit Planets
Reading - Mars (IMS & DB2)
Reading - Saturn (MQ & CICS)
Save Readings (EJB)

- 001
- Mars
- 77
- 11
- 07
- 2000-03-11
- 14:14:14

Figure 264. Topology 2: IMS result

CICS and MQSeries connector

The real PDK application ships with MQSeries, but CICS is simulated. For our purpose, we modified the code to return dummy data (Figure 265).

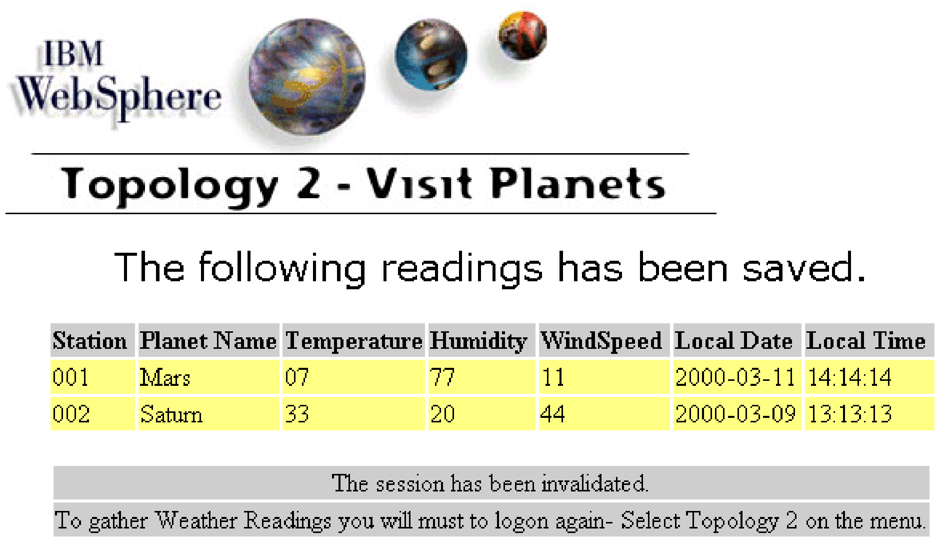

IBM WebSphere
Topology 2 - Visit Planets
Reading - Mars (IMS & DB2)
Reading - Saturn (MQ & CICS)
Save Readings (EJB)

Station	Planet Name	Temperature	Humidity	WindSpeed	Local Date	Local Time
002	Saturn	20	44	33	2000-03-09	13:13:13

Figure 265. Topology 2: CICS and MQSeries result

EJB

The real PDK application ships with an Enterprise JavaBean that stores the collected new weather readings in a DB2 table. For our purpose, we did not install the EJB, and we modified the code to not invoke the EJB (Figure 266).



The following readings has been saved.

Station	Planet Name	Temperature	Humidity	WindSpeed	Local Date	Local Time
001	Mars	07	77	11	2000-03-11	14:14:14
002	Saturn	33	20	44	2000-03-09	13:13:13

The session has been invalidated.

To gather Weather Readings you will must to logon again- Select Topology 2 on the menu.

Figure 266. Topology 2: EJB result

This is the full extent of the PDK application which has been implemented for this book.

12 Using Patterns for e-business to build the PDK

This chapter discusses the design of some of the applications contained in the Pattern Development Kit (PDK). In doing so, we provide a brief introduction to how you can use Patterns for e-business to build an e-business application.

We will go through the different types of Patterns for e-business, and then focus in on the pattern that is used in the PDK. In our explanation of the pattern, we will work through the different steps required in creating the overall application architecture used in our examples.

This chapter is not intended to be a complete guide to using Patterns for e-business. For more information, visit the Patterns for e-business Web site at `http://www.ibm.com/software/developer/web/patterns/`.

For detailed information about design patterns used in the PDK refer to *Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

Benefits of Patterns for e-business

Patterns for e-business can be used to assist you when designing and building an e-business application. They give you starting templates upon which to build your solution, saving you from having to architect your application from scratch. They can both speed up the design process and aid you in making sure that you have considered all relevant architectural tiers. By using Patterns for e-business, you can utilize the experience of others, while also customizing the solution to your own needs.

It is worth noting that these patterns should be used in conjunction with a proven development methodology to help ensure customer requirements are fully met.

Applying Patterns for e-business

When using Patterns for e-business, we apply a top-down approach. Starting at a higher, more abstract level, we decide on the type of business the application is for. We then take this pattern and begin to drill down through the levels until we reach the physical products that will underlay the application's actual implementation.

The steps involved are:

1. Choose a business pattern.
2. Choose a related logical pattern.
3. Choose a related physical pattern.
4. Design your solution.

We will now follow through these steps as if we were using them to create the PDK application. As a result, we will not show all the options available at each step. For the full set of expanded options, refer to the Patterns for e-business Web site.

Choose a business pattern

Business patterns are used to describe the interaction between the different participants in an e-business application. Those participants can be physical users, data, or businesses.

The following are some well-defined business patterns:

- ❑User-to-business
- ❑User-to-online buying
- ❑Business-to-business
- ❑User-to-data
- ❑User-to-user
- ❑Application integration

We will be looking more closely at the user-to-business pattern. For more information on the others, please refer to the Patterns for e-business Web site.

User-to-business pattern

This pattern is used to deal with cases of users (internal or external), interacting with existing enterprise transactions or data. It is commonly used in situations where the enterprise handles goods and services that are not normally listed in, or sold from a catalog. It basically covers all user-to-business transactions not covered by the User-to-Online Buying pattern.

Common business scenarios

Scenarios that easily fit under this pattern include:

- ❑Convenience banking
- ❑Discount brokerage (online share-trading)

Choose a related logical pattern

Once you have selected the appropriate business pattern for your scenario, you need to then look at the associated logical patterns and decide which best suits your application.

Logical patterns (or topologies) allow you to describe how the applications within your solution will interact. They also describe the runtime infrastructure required to deliver the necessary functionality.

There are two logical patterns you will need to make selections from:

- ❑Logical application topology
- ❑Logical runtime topology

Logical application topology

This topology is primarily focused on showing the shape of the application, its logic and associated data. They are not concerned with showing things like middleware, or file location.

The PDK utilizes the user-to-business application topologies 1 and 2. We will describe topology 2 in more depth, as topology 1 is really just a subset of this. However, if you refer back to the Patterns for e-business Web site, you will be able to see the other variations in full.

User-to-business application topology 2

This solution is referred to as Web-centric. While focusing on a clean separation between the presentation and application logic, it allows for one or more point-to-point connections to back-end legacy applications or databases (Figure 267).

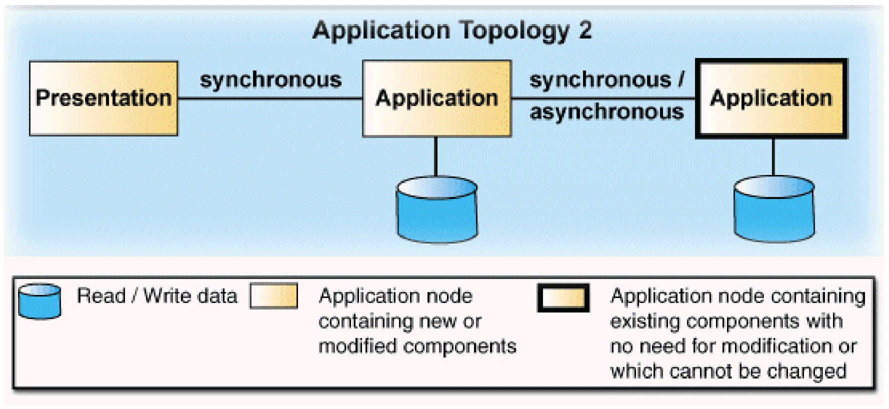

Application Topology 2
Presentation
synchronous
Application
synchronous / asynchronous
Application
Read / Write data
Application node containing new or modified components
Application node containing existing components with no need for modification or which cannot be changed

Figure 267. U2B application topology 2

Logical runtime topology

The runtime topology identifies the different nodes which are responsible for your functional requirements. It also begins to place those nodes in conceptual locations.

User-to-business runtime topology 2

This runtime topology (Figure 268) is used in conjunction with the user-to-business application topology 2.

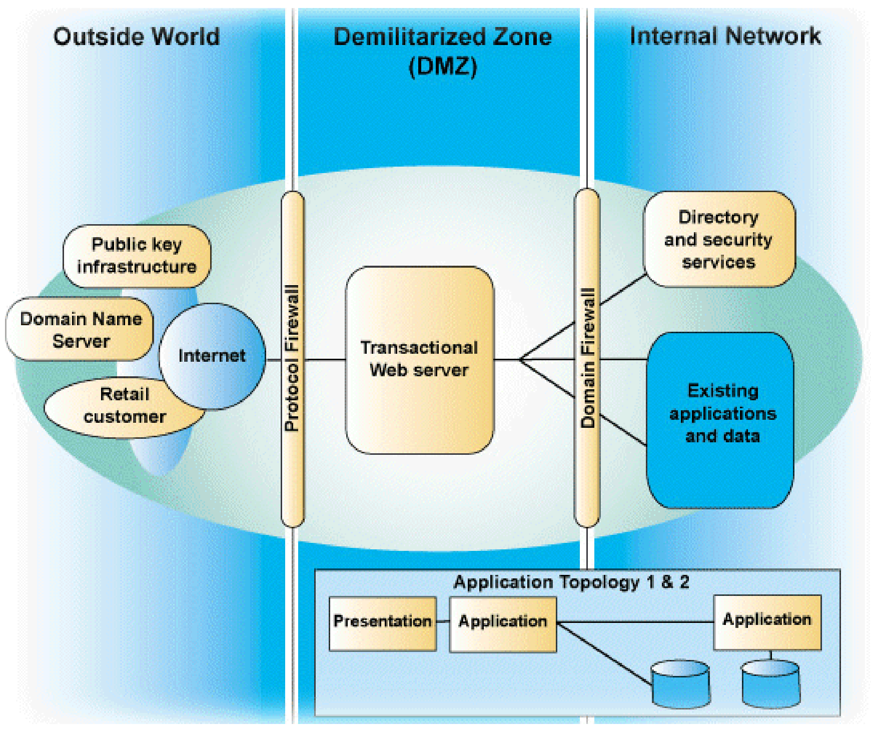


Figure 268. U2B runtime topology 2

Choose a related physical pattern

So far, what we have discussed has been platform-independent. It is now time to apply these patterns to a physical platform.

Applying the product mapping

For the purposes of the PDK, we apply the chosen topologies to the UNIX and NT platforms (Figure 269).

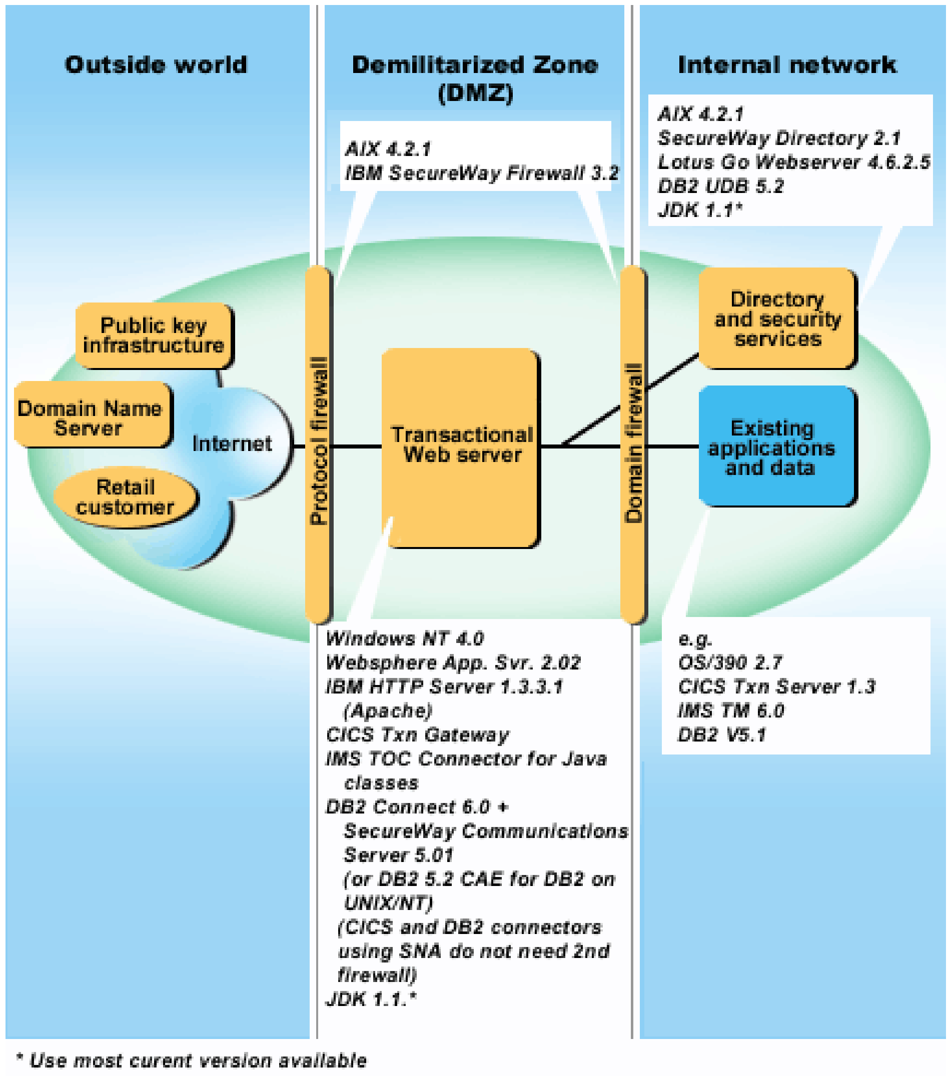

Outside world
Demilitarized Zone (DMZ)
Internal network
AIX 4.2.1
IBM SecureWay Firewall 3.2
AIX 4.2.1
SecureWay Directory 2.1
Lotus Go Webserver 4.6.2.5
DB2 UDB 5.2
JDK 1.1*
Public key infrastructure
Domain Name Server
Internet
Retail customer
Protocol firewall
Transactional Web server
Domain firewall
Directory and security services
Existing applications and data
Windows NT 4.0
Websphere App. Svr. 2.02
IBM HTTP Server 1.3.3.1 (Apache)
CICS Txn Gateway
IMS TOC Connector for Java classes
DB2 Connect 6.0 + SecureWay Communications Server 5.01 (or DB2 5.2 CAE for DB2 on UNIX/NT)
(CICS and DB2 connectors using SNA do not need 2nd firewall)
JDK 1.1.*
e.g.
OS/390 2.7
CICS Txn Server 1.3
IMS TM 6.0
DB2 V5.1
* Use most curent version available

Figure 269. U2B runtime topology 2 product mapping

Once again, variations upon this are available, and you should refer to the Patterns for e-business Web site for more information.

The next steps

We now have a good idea of the underlying architecture to our application. All that is left is to actually create it! We will not be going into detail about the design decisions involved in creating the PDK. However, in the next section ("Design techniques used"), we will discuss two of the techniques that are commonly used throughout.

Design techniques used

To provide robust and scalable software, the PDK has based its solution design around some universally-accepted design techniques. The two prominent ones are:

- ❑Model-View-Controller framework
- ❑Command design patterns

The Model-View-Controller framework

When designing an application, you first need to understand the structure of how your various components are going to interact. Taking the information from the Logical Patterns above and what we know about Web applications, we can begin to develop a general picture of how things may work (Figure 270).

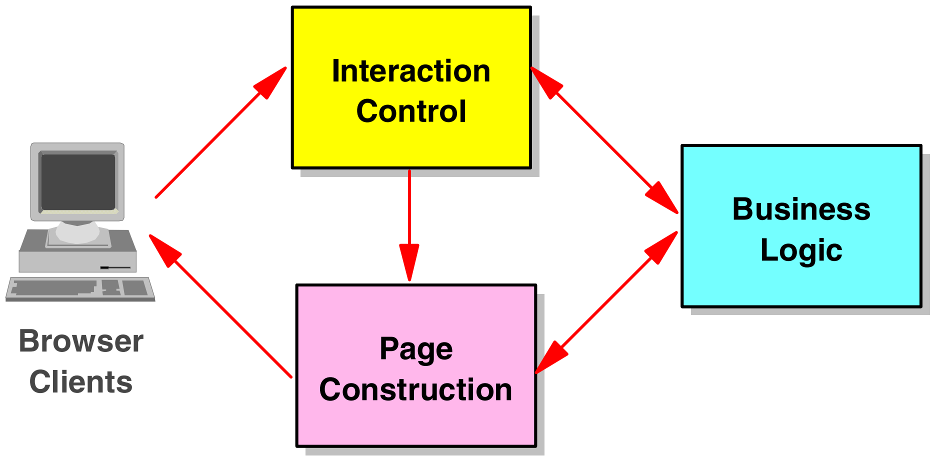


Figure 270. The structure of Web interactions

Those who are familiar with this structure will begin to see that it very closely resembles the well-known Model-View-Controller (MVC) framework. As its name suggests, it is made up of three components:

- ❑The model is responsible for the underlying data, and transactions that can be associated with it. This is the *business logic*.
- ❑The view is responsible for displaying the data. This is the *page construction*.
- ❑The controller is responsible for decoupling the interactions between the model and the outside world. This is the *interaction control* and need have no knowledge of how the View works.

The benefits of using MVC

This separation of business logic from the user interface allows you greater flexibility in your application. If your user interface (the Web pages) must change in look and feel, then the other segments of the application need not be heavily affected, if at all.

Also, the very fact that you can split your application into three fairly distinct sections, each requiring different skills, allows you to better manage your development cycle and team.

This separation into model, view, and controller sections can be clearly seen in the final designs of the PDK (see “The design for the PDK” on page 355).

The Command bean design pattern

Design patterns have been well-recognized for a number of years now, and they provide useful templates for solutions to common problems. For more information on the Command pattern and design patterns in general, refer to the book *“Design Patterns - Elements of Reusable Object-Oriented Software”, by Gamma, Helm, Johnson and Vlissides.*

The Command pattern is a type of behavioral design pattern, which means it is concerned with how objects relate to and communicate with each other.

Where is it used?

The Command pattern is useful in scenarios where it is necessary to issue requests to an object without knowing anything about the operation being requested or how that operation is carried out. For example, in a menu system each menu item triggers a command request, but the menu item need not know anything about the request, except for when it must be triggered.

How is it implemented?

This pattern is commonly implemented by making all your commands objects of a defined class (Command) which responds to an *execute* method. The object that requests the command merely needs to populate the command object with any required data, and then simply call the *execute* method. This enables the calling object to succeed in its task, while knowing little about the command it is actually calling.

What are the benefits

This pattern provides an extra level of decoupling within your application. As the calling object is protected from knowing how or what the command is, it is more robust should changes to this command be required. The command could achieve its task in a completely different way from how it was first envisaged (for example, retrieving data from a new database as opposed to a legacy application), and the calling object would not have to be changed. This provides you with a much more flexible product in the long term.

Implementing the command pattern will make things easier if you wished to implement a logging system for the commands, or even the ability to roll-back your commands in a clean and simple manner.

Also, in the shorter term, the use of the Command pattern can help you break up development into its separate components, and allow you to focus your team into specific areas.

The design for the PDK

The PDK combines the patterns and techniques described above into a real application. To give you a clearer understanding of how this works, we will now break down the design of topology 1 and topology 2. These are the two parts of PDK that are included with this book.

To make full use of the following descriptions and illustrations, you should use them in combination with Chapter 11, “Pattern Development Kit overview” on page 337, which walks you through the running of topology 1 and 2.

Topology 1

The part of the PDK referred to as topology 1 is responsible for doing a database query based on user input, and displaying the result back to the user. Also, it carries out a database update, storing the user’s query.

Stage A

Figure 271 displays the component flow of topology 1. In particular, it shows how the Command pattern is used.

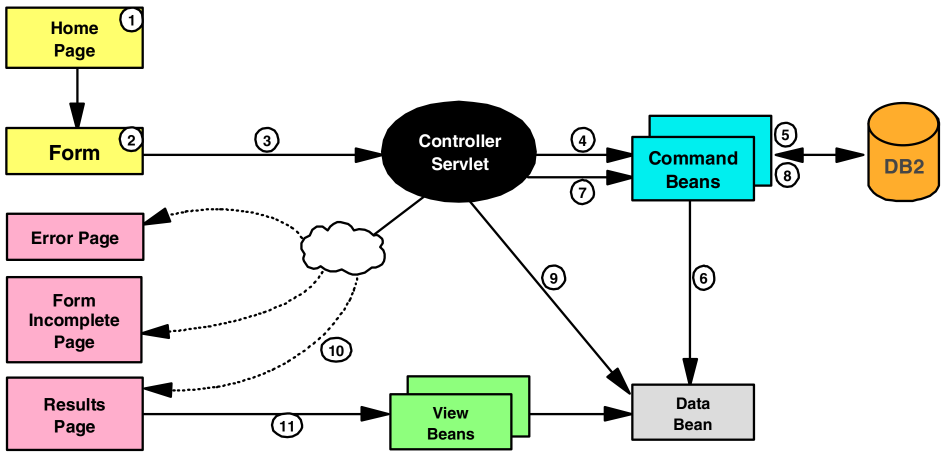


Figure 271. Topology 1 component flow: stage A

Interaction steps

1. The user selects *Topology 1* from the home page.
2. The Web server returns a JSP form requesting some input parameters.
3. The form data is posted to the controller servlet which validates it.
4. The controller servlet instantiates the command bean responsible for retrieving the search results and sets its properties with the appropriate data.
5. The controller servlet calls the command bean's *execute* method.
6. The command bean retrieves historical data from the database and stores the results in a historical data bean.
7. The controller servlet initializes and sets the properties for a second command bean responsible for storing the user's query request in a journal table.
8. The controller servlet calls the second command bean's *execute* method.
9. The servlet stores the historical data bean on the request object.
10. The servlet invokes a JSP to display the results. The servlet determines if the browser can display XML and depending on the result, a different JSP is used.

11. The JSP uses a related view bean to retrieve the historical data from the request object, and to format it appropriately for output.

Topology 2

Topology 2 consists of a multiple stage process. The first stage is responsible for authenticating the user via LDAP using SecureWay Directory, and the second stage retrieves available menu options from the database, based on that user's employee type. The steps that follow retrieve data from enterprise resources.

Stage B

Figure 272 displays the component flow of topology 2 stage B.

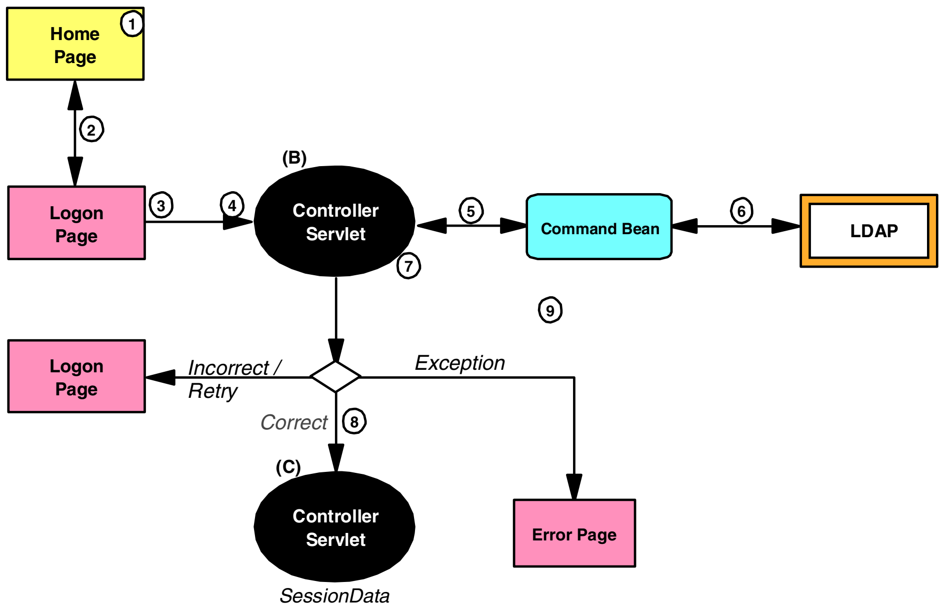


Figure 272. Topology 2 component flow: stage B

Interaction steps

1. The user selects *Topology 2* from the home page.
2. An SSL connection is established between the user's browser and the Web server.
3. The user is presented with a HTML logon form requesting a user name and password.

4. The user completes the form and posts it to the controller servlet (B).
5. The controller servlet instantiates a command bean which is responsible for authenticating the user based on the form data.
6. The command bean binds to LDAP to authenticate the user.
7. The servlet then retrieves the employee type for that user and stores it as session data.
8. After the user is authenticated, a second controller servlet (C) responsible for retrieving the list of menu options is called by the first controller servlet. The list returned is based on the user's employee type.

Stage C

Figure 273 displays the component flow of topology 2 stage C.

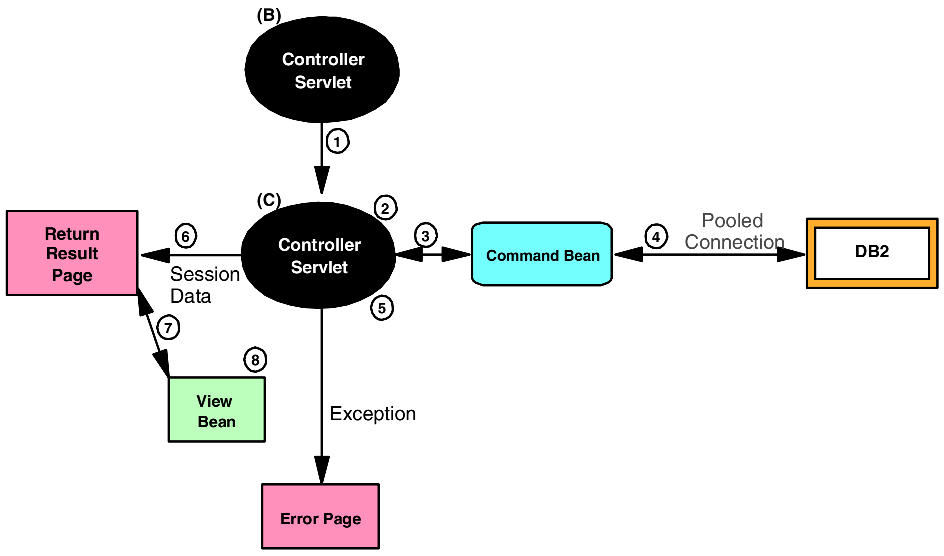


Figure 273. Topology 2 component flow: stage C

Interaction steps

1. The controller servlet (C) in this interaction is called by the controller servlet (B) in the previous step.
2. The servlet retrieves the user's profile (the employee type) from the session.
3. The servlet instantiates a command bean, set the properties and invokes the execute method. The command bean is cached in the session object.

4. The command bean, in its execute method, obtains a DB2 connection from the pool and queries the database for the required data.
5. The servlet stores the menu options returned on the request object.
6. The servlet then calls the output JSP to display the results.
7. The JSP interrogates the view bean to display the results to the user.
8. The view bean retrieves the menu options from the request object, and formats them appropriately.

Stages D through F retrieve data from enterprise resources. We do not describe these in as much detail as the previous stages; rather, we just show an overview diagram for each stage.

Stage D

Stage D (Figure 274) interacts with CICS and MQSeries systems.

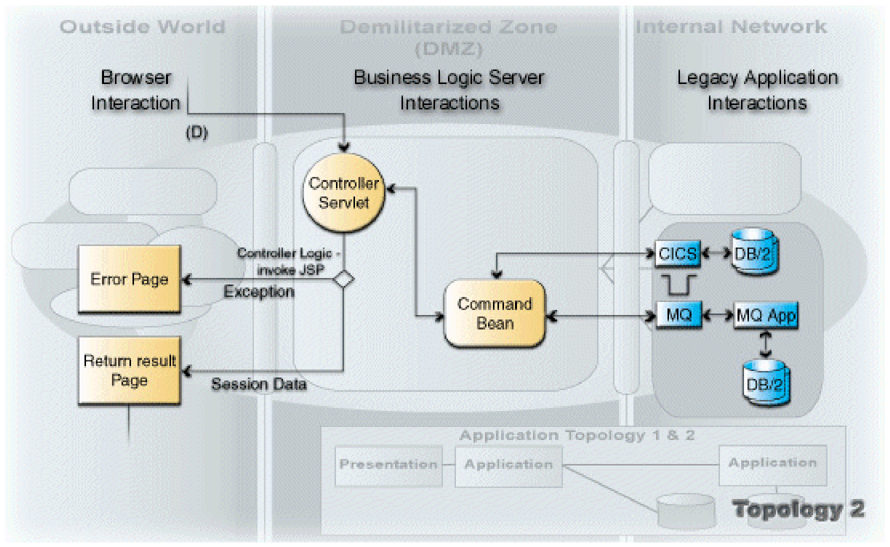


Figure 274. Topology 2 component flow: stage D

Stage E

Stage E (Figure 275) interacts with IMS and DB2 systems.

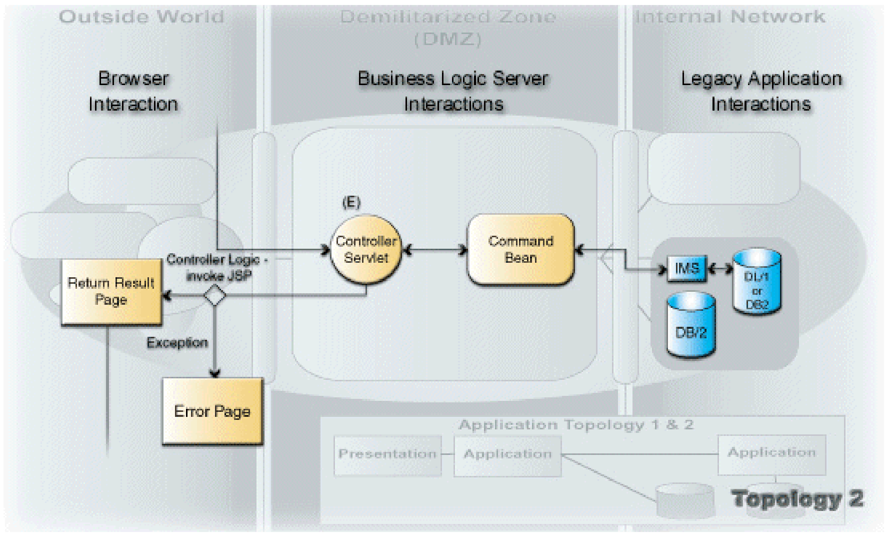


Figure 275. Topology 2 component flow: stage E

Stage F

Stage F (Figure 276) interacts with DB2 through an Enterprise JavaBean.

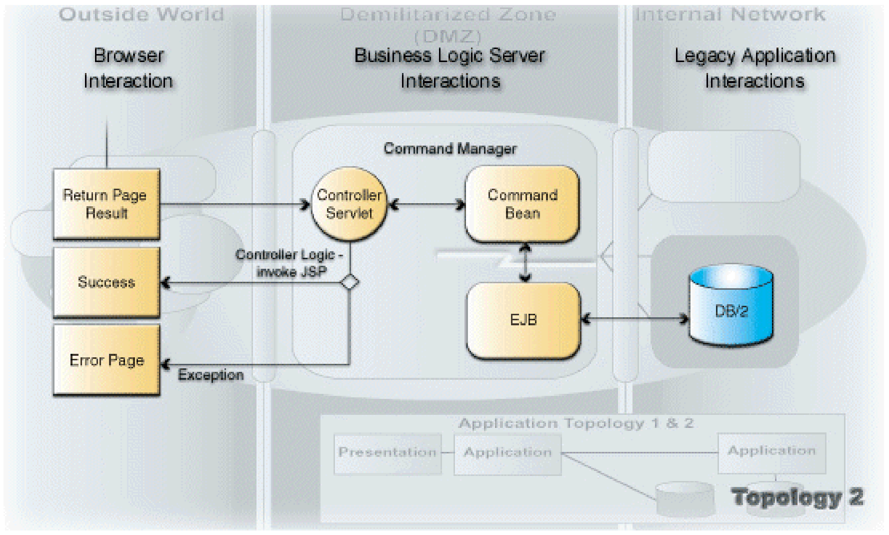


Figure 276. Topology 2 component flow: stage F

In Summary

We have now discussed the design used in the PDK, and walked through the different steps that make up some of its applications.

For more information on the PDK, and for the full set of applications it contains, please refer to *Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

For more information on Patterns for e-business, refer to the Patterns for e-business Web site:

`http://www.ibm.com/software/developer/web/patterns/`

13 Running the PDK in WebSphere

This chapter describes how to install the Pattern Development Kit application for use under WebSphere Application Server and IBM HTTP Server.

Much of the installation process has been automated by using CMD files. While it would have been possible to merge these CMD files into a more seamless process, we have chosen to break out the installation steps as much as possible, thus allowing us to discuss the details of each step.

The installation instructions provided in this chapter are intended to be followed sequentially.

Extracting the resources

The Pattern Development Kit source files are provided in the `5755pdk.zip` file. Extract the file to the root directory of your hard disk. When extracted, your top-level directory structure should look like this:

```
d:\sg245755\pdk\...
```

All scripts (.cmd files) to configure the application can be found in the `\pdk\cmd` directory. All scripts to reset the changes made to configure the application can be found in the `\pdk\cmdReset` directory.

To run a script, double-click on the file in the Windows NT Explorer.

Tailoring the installation system

There are a number of configuration steps that you have to complete before executing further scripts in the installation process. The steps in this section make the required modifications to the XML configuration files used by WebSphere Application Server to support the sample code.

User ID

The DB2 database is accessed through the user ID *USERID* with password *password* (in lower case). Define such a user ID in your Windows NT system.

Set up environment parameters

The first script that you have to edit sets up environment variables used by other scripts during the installation process.

Open the `itsoEnv.cmd` file in a text editor. Configure each line in this file according to the product installation directories on your computer. A typical configuration may look like this:

```
set NODENAME=fundy
set IBMHTTPSERV=C:\Program Files\IBM HTTP Server
set IBMKEYMAN=C:\Program Files\IBM\GSK
set WSAPPSERV=C:\WebSphere\AppServer
set VAJAVARES=C:\IBMVJava\ide\project_resources\
              IBM WebSphere Test Environment
set ITSOTOPO=D:\SG245755\Pdk
```

The *nodename* variable is case-sensitive, so make sure that you enter it in lowercase, identical to the WebSphere Application Server topology view.

Be careful and verify for each product that the information entered in this step is correct.

Tailor the XML files

Now you have to tailor the XML files that are used to configure the application server.

Run the script `tailorXMLfiles.cmd`.

This script uses the Notepad editor to display a number of XML configuration files that you have to update with information specific to your computer setup. Save the changes for each edited file.

Edit db2jdbcdriver.xml

The `db2jdbcdriver.xml` file defines DB2 driver information used by WebSphere data sources for the PDK.

Replace the Xs with the host name that you specified in the `itsoEnv.cmd` file, ensuring that it is in lower case:

```
<node-name>XXXXXXXX</node-name>
```

Also, change the following line to reflect the correct path to the `db2java.zip` file on your computer.

```
<jdbc-zipfile-location>C:\SQLLIB\java\db2java.zip</jdbc-zipfile-location>
```

Edit securehost.xml

The `securehost.xml` file defines a second host, in addition to the default_host.

Toward the end of this file, you need to specify the host aliases for your computer by replacing the lines marked by X with appropriate values. The configuration specified in this file is used when a secure request is made via HTTPS in the topology two example.

```
<alias-list>
   <alias>xxxxxxxx.almaden.ibm.com:443</alias>
   <alias>xxxxxxxx:443</alias>
   <alias>xxx.xxx.xxx.xxx:443</alias>
   <alias>127.0.0.1:443</alias>
   <alias>localhost:443</alias>
</alias-list>
```

In the first two aliases, use the computer host name entered in the `itsoEnv.cmd` file. In the third alias, specify the IP address of the host computer. To obtain the IP address, type the following in a command prompt on the host computer (the host is our local machine in this case):

```
ping <hostname>
```

You should be presented with a number of lines of return data from the `ping` command as shown below. Enter this IP address into the third alias line.

```
Reply from 9.1.151.36: bytes=32 time<10ms TTL=128
```

The default port used by the HTTPS protocol is :443. You should not have to change this value.

Edit webapptopologyone.xml

The `webapptopolgyone.xml` file is responsible for updating the WebSphere Application Server to reflect the correct paths to the source code and HTML/JSP files required by the *topologyone* Web application. These changes are necessary, as the topology application runs under Web applications called *topologyone* and *topologytwo*, rather than running under the *default_app* Web application provided by default in WebSphere Application Server.

The first change requires that you specify the computer host name in the line:

```
<node name="XXXXXXXX" action="update">
```

Next, locate the lines specified below and change the paths to reflect the correct path on your machine. This tells the application server where to find HTML/JSP pages and class files for the application.

```
<document-root>C:\WebSphere\AppServer\hosts\default_host\topologyone\web
</document-root>
...
<classpath>
<path
   value="C:/WebSphere/AppServer/hosts/default_host/topologyone/servlets"/>
</classpath>
...
```

Locate the text block beginning with:

```
<servlet name="JSP 1.0 Processor" action="update">
```

Within this block, find the following lines and change the `value` attribute of the `<parameter name>` tag to the correct path for your machine. This updates the path used by the JSP engine to locate and compile the applications’ JSPs:

```
<init-parameters>
   <parameter name="workingDir"
      value="C:\WebSphere\AppServer\hosts\default_host\topologyone\web"/>
   </init-parameters>
```

Edit webapptopologytwo.xml

Repeat the instructions for editing `webapptopologyone.xml`.

Edit start.xml, stop.xml, restart.xml, reset.xml

In these four files, enter the correct host name in the line:

```
<node name="XXXXXXXX" action="update">
```

Installing and running the Pattern Development Kit

In the following steps, we will configure the application resources and database in addition to configuring the IBM HTTP Server and SecureWay LDAP directory services required by the Pattern Development Kit example.

Restart the HTTP Server

The first script you run stops and then restarts the IBM HTTP Server service:

```
RestartHttpServer.cmd
```

Create a self-signed SSL certificate

This step creates a new SSL certificate which allows the HTTP Server to perform encrypted communication over HTTPS. The `startIBMKeyManagementUtility.cmd` script executes the key management software provided with the IBM HTTP Server to create and manage the certificate.

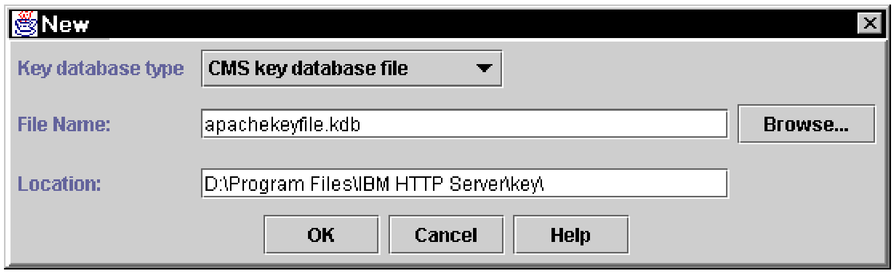
First, you have to create the `\key` directory underneath the `\IBM HTTP Server\` directory to store the key created in the next step:

```
createSSLKeyDirectory.cmd
```

Next, you run a script to start the IBM key management software that enables you to create the certificate:

```
startIBMKeyManagementUtility.cmd
```

In the IBM Key Management window, select *Key Database File -> New* to commence the certificate creation process. The *New* dialog appears (Figure 277). Click the *Browse* button and locate the `d:\..\IBM HTTP Server\key` directory. Enter `apachekeyfile.kdb` in the Key Database File Dialog and click the *Save* button.

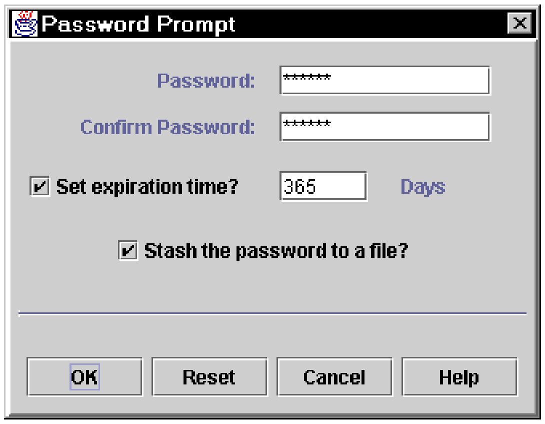


Key database type: CMS key database file
File Name: apachekeyfile.kdb
Location: D:\Program Files\IBM HTTP Server\key\

Figure 277. Creating a new key file

Click *OK* in the *New* dialog to display the *Password Prompt* dialog (Figure 278). This dialog allows you to enter a password for the Key file. Enter the same password that you have specified in your WebSphere Application Server and DB2 configurations for your administrative logon name.

Select the options as shown in Figure 278 and click *OK*. You should receive a prompt alerting you that the password has been saved.



Password: ******
Confirm Password: ******
☑ Set expiration time? 365 Days
☑ Stash the password to a file?

Figure 278. Setting options in the Password Prompt dialog

In the main IBM Key Management window, click the drop-down list titled *Signer Certificates* and select *Personal Certificates* from the list. Click the *New Self Signed...* button to display the *Create New Self-Signed Certificate* dialog. Enter the following values for the specified fields:

❑In the Key Label field enter, `User-to-Business Design Pattern Certificate`.

❑In the Version field, ensure that `X509 V3` is selected.

❑In the Key Size field, select `512`.

❑In the Common Name field, enter the fully qualified host name of your computer.

❑In the Organization field, enter `IBM Hursley`.

❑In the Organization Unit field, enter `ASG`.

❑In the Country field, ensure the selected country is `US`.

❑In the Validity Period field, ensure the selected period is `365`.

Click the *OK* button in the *Create New Self-Signed Certificate* dialog.

You will notice that the certificate is added to this list of certificates in the *Personal Certificates* category in the main window. You can now exit the IBM Key Management utility by selecting *Key Database File -> Exit*.

Create the Web site

This step creates the directory structure under the `\IBM HTTP Server\htdocs\` directory and populates these directories with core resources required by the PDK applications:

```
createSkeletonWebSite.cmd
```

The PSK application uses this directory setup:

```
\IBM HTTP Server\htdocs\U2BTop\
                      \U2BTop\images
                      \U2BTop\theme
```

Configure IBM HTTP Server

The next step modifies the `httpd.conf` file of the IBM HTTP Server using the values you provide in this step:

```
createSkeletonConfig.cmd
```

The `changes.conf` file is displayed.

Edit changes.conf

Locate the following line and modify it to reflect your computer's host name:

```
ServerName yourHostname
```

Locate lines that contain the path of the IBM HTTP Server and change the path to your installation:

```
LoadModule ibm_ssl_module "C:/IBM HTTP Server/modules/IBMModuleSSL128.dll"
<Directory "C:/IBM HTTP Server/htdocs/u2btop" >
Keyfile "C:/IBM HTTP Server/key/apachekeyfile.kbd"
```

Note: Outside the USA you may have to use the `IBMModuleSSL56.dll`.

Check http.conf

The changes.conf file is appended to the http.conf file of the HTTP server and then displayed to you. Verify that the changes made to the `http.conf` file are correct. In particular, locate the comment block below and check the subsequent line entries to ensure correct server name and path values are correctly specified.

```
#
#===================================================================
# changes to IBM HTTP Server\conf\http.cnf for User-to-Business Patterns
```

Restart the IBM Http Server

Following these changes, you must restart by IBM HTTP Server to activate the new configuration:

```
restartHttpServer.cmd
```

Quick test of HTTP Server configuration

To quickly test if the configuration is successful at this point, run the script appropriate for your Web browser.

For Internet Explorer users, run: `startIE5.cmd`

For Netscape Navigator users, run: `startNetscape.cmd`

LDAP configuration

The Pattern Development Kit authenticates users via LDAP directory services. This step sets up the LDAP services provided with IBM SecureWay to enable this authentication process to function correctly.

If you have not already installed IBM SecureWay Directory, refer to the Product Installation chapter for information about this product. At the time of writing, IBM SecureWay Directory 3.1.1 does not work with DB2 Fixpack 2 and requires DB2 Fixpack 1A, but Version 3.1.1.5 does work.

Start the LDAP directory server

To start the LDAP directory server, run the script:

```
startLDAPServer.cmd
```

This will start the IBM SecureWay Directory V3.1 service. When it has started successfully, display the LDAP configuration panel by launching your Web browser and entering the URL:

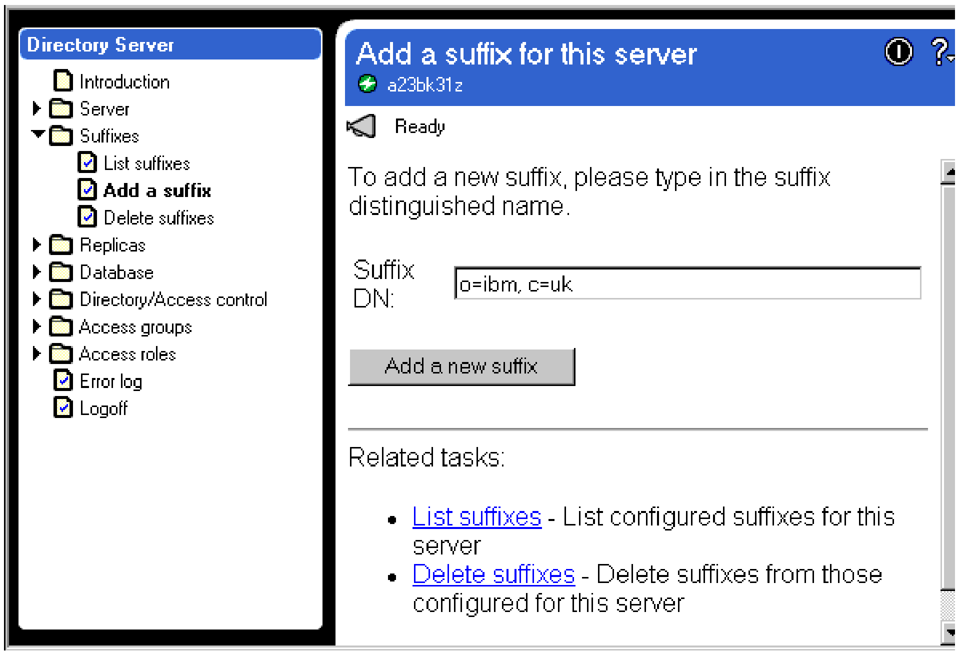
```
http://<YourHostName>/ldap
```

In the Logon panel, enter the following values:

❑In the User ID field enter, `cn=youruserid`

❑In the Password field, enter, `yourpassword`

Click the *Logon* button to display the LDAP configuration options. Select *Add a Suffix* from the *Suffix* node in the tree (Figure 279).



Add a suffix for this server

a23bk31z

Ready

To add a new suffix, please type in the suffix distinguished name.

Suffix DN: o=ibm, c=uk

Related tasks:

- List suffixes - List configured suffixes for this server
- Delete suffixes - Delete suffixes from those configured for this server

Figure 279. Adding a new suffix to LDAP

Enter the following text in the Suffix DN field:

```
o=ibm, c=uk
```

Click the *Add a new suffix* button to confirm the new suffix definitions. Next, from the following confirmation screen, select the `restart the server` link.

Import the LDIF file

The final step for LDAP configuration is to import the LDIF file. This file updates the LDAP directory with user information required by the application:

```
importLDIFfile.cmd
```

Create the ITSOTOPO database

To create the DB2 database with all the tables used by the application, run:

```
createDatabase.cmd
```

This script launches the DB2 command-line processor and issues the CREATE DATABASE command. Afterwards the tables are defined and loaded with initial data. This process may take some time.

You can also reload the tables using the script:

```
reloadDatabase.cmd
```

Copy application-specific files

Now you have to create the Web application directories under the `default_host` and `secure_host` directories of the WebSphere Application Server:

```
createWebAppDirectories.cmd
```

The directories created by this step are:

```
d:\WebSphere\AppServer\hosts\default_host\topologyone
d:\WebSphere\AppServer\hosts\secure_host\topologytwo

%WSAPPSERV% = d:\WebSphere\AppServer      (in itsoenv.cmd)
```

Next, you copy the servlet classes and JSPs required by each application step to the `\web` and `\servlet` directories for each host:

```
copyWebAppFiles.cmd
```

Import the XML configurations into WebSphere

To complete the installation, you run the scripts that import the XML configuration files generated by previous steps into the WebSphere Application Server.

First, you start the WebSphere Application Server service, if it is not running:

```
startWebSphereAdmin.cmd
```

Next, you start (or restart) the default server in WebSphere Application Server. The process of initializing the default server in WebSphere may take some time, so allow a minute or two after executing this step before moving on to subsequent steps:

```
startWebSphereServer.cmd
```

Now, you run the scripts responsible for importing the XML files that configure the WebSphere Application Server to run the example applications. Follow this sequence:

```
wasJDBCDriver.cmd
wasDataSources.cmd
wasVirtualHost.cmd
wasWebAppOne.cmd
wasWebAppTwo.cmd
```

Restart the default server node in WebSphere Application Server to enable the changes made by the previous scripts:

```
startWebSphereServer.cmd
```

Now, you start the WebSphere Administration Console:

```
startWebSphereClient.cmd
```

You should be able to expand the *Topology* in the Administration Console to view the configuration changes that have been made during this installation process.

Run the application

The installation is complete. You can run the Pattern Development Kit application by executing the script appropriate for your browser:

```
startIE5.cmd
startNetscape.cmd
```

You should see the main page as shown in Figure 257 on page 339. Click the *Continue* image to start using the application.

Follow the application as described in Chapter 11, “Pattern Development Kit overview” on page 337.

Resetting changes

We have included a number of script files that reset the configuration changes made during the setup of the Pattern Development Kit example. Run these scripts if you want to undo the changes or if you have made a mistake and want to restart the installation process:

```
   resetHTTPServer.cmd          <== reset the IBM HTTP Server configuration
   resetWebSphere.cmd           <== reset the WebSphere configuration
   resetDatabase.cmd            <== remove the ITSOTOPO database from DB2
```

14 Running the PDK in VisualAge for Java

In this chapter, we describe how to install the Pattern Development Kit to run under the WebSphere Test Environment in Visual Age for Java.

This chapter assumes that you have completed the installation for the WebSphere Application Server detailed in Chapter 13, “Running the PDK in WebSphere” on page 363. As a minimum, the ITSOTOPO database must be created and loaded with data.

Most of the configuration is performed to files in directories in the WebSphere Test Environment path, which is usually:

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\...
```

Where appropriate, we use the abbreviation of <WTE> to indicate this path.

Automatic configuration

You can perform an automatic update of the necessary configuration files for this application. Using the automatic configuration is the easiest way to configure the Pattern Development Kit to run in the WebSphere Test Environment. If you do not want to automate this process, you could make all changes manually as described in “Manual configuration” on page 378.

Running the configuration script

Provided in the `\Pdk\Cmd` directory is the script that performs most of the WebSphere Test Environment configuration:

```
setupVaJava.cmd
```

The functions performed by this script are:

- ❑Back up the `default.servlet_engine` file as `default.servlet_engine.save` in the <WTE> directory.
- ❑Create the following directories under the WebSphere Test Environment:

```
\hosts\default_host\topologyone\web
\hosts\default_host\topologyone\servlets
\hosts\default_host\topologytwo\web
\hosts\default_host\topologytwo\servlets
\temp\Jsp1_0\topologyone
\temp\Jsp1_0\topologytwo
```

- ❑Copy the preconfigured `default.servlet_engine` file from the `\Pdk\VaJava` directory to the <WTE> directory. The new file contains *web-group* configuration information for the two topology Web applications.
- ❑Copy preconfigured `.webapp` files provided with the kit to the `\servlets` directory for each Web application.:

```
<WTE>\hosts\default_host\topologyone\servlets\topologyone.webapp
<WTE>\hosts\default_host\topologytwo\servlets\topologytwo.webapp
```

- ❑Copy the `error.jsp` page to the web subdirectory of each Web application.
- ❑Copy the SnoopServlet.class to the servlets subdirectory for topology 2.
- ❑Copy the HTML files and images from the PDK to the default_app application:

```
\hosts\default_host\default_app\web\U2BTop\...
```

- ❑Copy the JSPs from the PDK to the Web application’s web subdirectory.
- ❑Copy the `.servlet` files used in *topologytwo* to the <WTE> directory.

Prepare a project and import the Java code

Start VisualAge for Java and create a new project named *ITSO Pattern Development Kit*.

Load the following features into the Workbench:

❑IBM WebSphere Test Environment
❑IBM Enterprise Access Builder Library
❑Connector HOD
❑Connector IMS TOC
❑Connector MQSeries
❑IBM Common Connector Framework
❑IBM Java Record Library

Import the Java source files from the two directories:

```
\SG245755\Pdk\Was\topologyone\servlets
\SG245755\Pdk\Was\topologytwo\servlets
```

Alternatively, you can import the *5755pdk.dat* repository file and then load the project or the packages into the Workbench.

Servlet engine configuration

The steps in this section configure the servlet engine in the WebSphere Test Environment. If you have chosen to do an automatic installation, you should complete these steps after running the `setupVaJava.cmd` script.

This setup is described in “WebSphere Test Environment — multiple Web applications” on page 215, and is repeated here in abbreviated format.

Modify ServletEngine properties

The first step is to modify the properties of the `ServletEngine` class in the WebSphere Test Environment. This step configures the ServletEngine class with class path and IDE version information and tells the ServletEngine which path it should consider as its application *server root* directory.

Locate the following class in the Visual Age for Java workbench:

```
com.ibm.servlet.engine.ServletEngine
```

In the properties dialog for this class, make the following changes:

❑In the *Program* tab edit the command-line field to read:

```
-serverRoot "d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment"
```

- ❑Edit the Properties field by adding the property:

```
ivj.version=3.02
```

- ❑In the *Class Path* tab edit the *Extra directories path* and copy all the entries from the com.ibm.servlet.SERunner class. (See Figure 164 on page 218.)
- ❑Edit the *Project path* and select Connector IMS TOC, Connector MQSeries, IBM Common Connector Framework, IBM Enterprise Access Builder Library, IBM Java Record Library, and the ITSO Pattern Development Kit.

Manual configuration

If you want to configure the WebSphere Test Environment manually to gain a more thorough understanding of the setup process, follow the manual configuration steps provided in this section.

First, complete the steps detailed in “Servlet engine configuration” on page 377 to configure the servlet engine. Then move on to complete the remaining steps in this section.

The files for manual configuration are contained in the `\Pdk\VaJava` directory.

Configure the Web applications

The PDK consists of two Web applications that must be configured in VisualAge for Java as described in “WebSphere Test Environment — multiple Web applications” on page 215. The steps are:

- ❑Update the `default.servlet_engine` file and add two <websphere-webgroups> for the two Web applications.
- ❑Create directories for the two Web applications:

```
<WTE>\hosts\default_host\topologyXXX\servlets
<WTE>\hosts\default_host\topologyXXX\web
<WTE>\temp\Jsp1_0\topologyXXX
```

- ❑Copy the error.jsp file into the web subdirectory of each application.
- ❑Copy the HTML and image files to the default application web subdirectory:

```
<WTE>\hosts\default_host\default_app\web\U2BTop
```

- ❑Copy the JSPs to the Web application web subdirectory:

```
<WTE>\hosts\default_host\topologyXXX\web
```

❑Copy .servlet files to the <WTE> directory.

❑Copy the topologyXXX.webapp configuration files to the servlets subdirectories of each application. One example is shown in Figure 280.

```
<?xml version="1.0"?>
<webapp>
   <name>topologyone</name>
   <description>Pattern Development Kit Topology One</description>
   <error-page>/ErrorReporter</error-page>
<servlet>
   <name>histData</name>
   <description>Topology One Historical Data Servlet</description>
   <code>com.ibm.hursley.asg.ws.skeleton.topologyone.sectiona.
                                    RetrieveHistoricalDataServlet</code>
   <servlet-path>/histData</servlet-path>
   <autostart>false</autostart>
   </servlet>
<servlet>
      <name>ErrorReporter</name>
       ......
   </servlet>
<servlet>
      <name>invoker</name>
       ......
   </servlet>
<servlet>
      <name>jsp</name>
       ......
      <init-parameter>
         <name>scratchdir</name>
         <value>$server_root$/temp/JSP1_0/topologyone</value>
      </init-parameter>
       ......
   </servlet>
<servlet>
      <name>file</name>
       ......
   </servlet>
</webapp>
```

Figure 280. Web application configuration file

Running the application

Because we configured multiple Web applications, we have to use the ServletEngine (and not SERunner) to start the WebSphere Test Environment.

Start the ServletEngine

Run the ServletEngine class to start the WebSphere Test Environment. Check the Console window to ensure that the Web applications are loaded successfully.

To run the application, enter the following URL in a browser:

```
http://localhost:8080/U2BTop/indexVAJ.html
```

We have two copies of the HTML files because the VisualAge for Java Test Environment does not support the *HTTPS* protocol for the topology 2 application.

Running without SecureWay LDAP

To run topology 2 without an active LDAP server, you can modify the *SecurityServlet* class (com.ibm.hursley.asg.ws.skeleton.topologytwo.sectionb).

If logon fails, a CommandException is thrown and an error page is displayed. To bypass the exception, locate the *performTask* method and scroll to the bottom. You should find this code fragment:

```
} catch (CommandException e) {
   /*
    * Call the logonError page - inform the user
    */
   getServletConfig().getServletContext().getRequestDispatcher
                     ("sectionB/logonError.jsp").forward(req, res);

   // if SecureWay is not installed
   // comment out above statement and use code below

   //String employeeType = "1";
   //boolean create = true;
   //HttpSession session = req.getSession(create);
   //session.putValue("skeleton.userType", employeeType);
   //RequestDispatcher rd = getServletContext().getRequestDispatcher
                                                      ("/menuOptions");
   //rd.forward(req, res);
}
```

Deactivate the call to the logonError.jsp and uncomment the code that simulates a successful logon. This enables you to get to the remaining functions of the application.

Resetting changes

To reset any configuration changes made by the installation process, run:

```
resetVaJava.cmd
```

You should only run this script if you have performed an automatic installation. If you have performed a manual installation, you must manually reverse the change made to the configuration.

15 Developing the PDK using WebSphere Studio

This chapter describes how to build topology 1 and topology 2 of the Pattern Development Kit in the WebSphere Studio environment, so that it can be further extended to include additional functionality.

We show how to build the PDK application as a project in the WebSphere Studio environment, how to integrate with VisualAge for Java, and how to deploy changes to the WebSphere Application Server environment.

Overview

The Pattern Development Kit (PDK) topology 1 and topology 2 examples included with this book are structured to deploy directly into the WebSphere Application Server environment. The step-by-step instructions in Chapter 13, “Running the PDK in WebSphere” on page 363, provide the information on how to deploy directly to the WAS environment, through executing a series of scripts which build and configure the PDK in the WAS environment.

The PDK is meant to provide useful examples of fully functioning Web applications that demonstrate important architectural and design concepts, in addition to important servlet interaction techniques that we discussed in Chapter 4, “Servlets” on page 41. The PDK is also meant to be extensible, in that it can be customized to add additional functionality, and perhaps provide a framework for a possible solution.

In this chapter, we show you how to load these topology examples into the WebSphere Studio environment, so that you can further extend the code to test out other configurations that you may want. We describe two scenarios, including one where we integrate with VisualAge for Java.

We recognize that this looks like reverse engineering. We do not necessarily endorse this as a common development process; however, you may find the information in this chapter useful when you want to port a Web application into WebSphere Studio for future development and management.

You can follow the steps outlined here to gain experience with setting up a WebSphere Studio project, or you can work with the Studio archive file *ITSO Pattern Development Kit.war* that we provide in `Sg245755\pdk\studio`. See “Opening an archive” on page 293 on how to work with an archive file to preserve the publishing targets.

Building the WebSphere Studio project

This section describes the configuration of the PDK in the WebSphere Studio environment.

Creating the WebSphere Studio project

Create a new project in WebSphere Studio named *ITSO Pattern Development Kit*. By default, you get *servlet* and *theme* folders. The final state is shown in Figure 281.

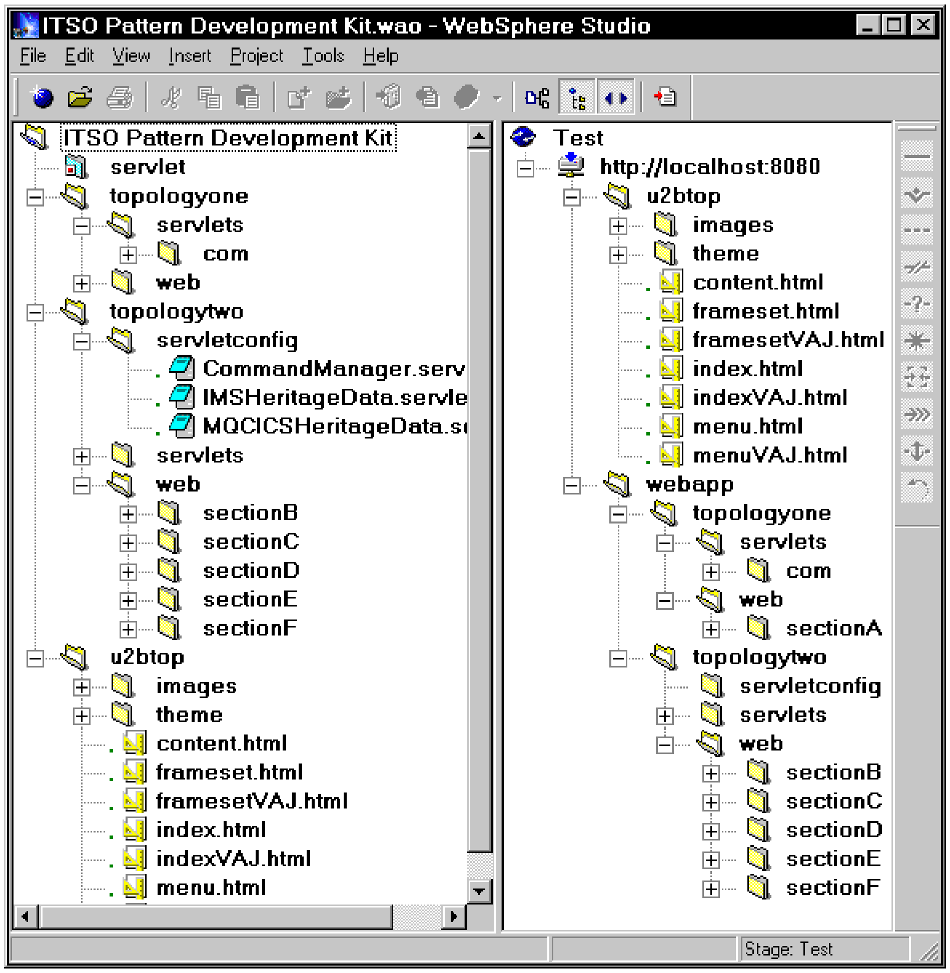


Figure 281. Pattern Development Kit WebSphere Studio project

Create the project folders

All the base code is available in subdirectories *was* and *website* of `d:\SG245755\pdk`. Here are the steps to arrive at the project layout. Use the *Insert -> File* and *Insert -> Folder* menu options and click on the *Use existing* tab:

❑Insert a new folder named *u2btop*.

❑Insert the HTML files from *website\htdocs* into the *u2btop* folder.

❑Insert a new folder under *u2btop* (use existing) and select *website\htdocs\images*.

❑Insert a new folder under *u2btop* (use existing) and select *website\htdocs\theme*.

❑Insert a new folder (use existing) and select *was\topologyone*.

❑Insert a new folder (use existing) and select *was\topologytwo*.

❑Insert a new folder named *servletconfig* under *topologytwo*. We want to publish the .servlet files separately from the servlets. Move the three .servlet files from topologytwo\servlets to topologytwo\servletconfig.

❑Remove the *theme* folder in the project, we have one under *u2btop*.

This completes the project layout, that is, the left half of Figure 281.

Define the publishing stages

We want to publish to VisualAge for Java (WebSphere Test Environment) and to WebSphere Application Server; therefore we require two publishing stages.

Create the Test publishing stage

We use the Test stage as the publishing view for the VisualAge for Java WebSphere Test Environment. Select the Test publishing view (*Project -> Publishing Stage -> Test*).

❑The default server is http://localhost. We want the Test stage for VisualAge for Java. Insert a new server (*Insert -> Server*) called *http://localhost:8080*.

❑Move all the existing folders in the publishing view from localhost to localhost:8080, and delete the localhost server.

❑Insert a new folder named *webapp*.

❑Move the *topologyone* and *topologytwo* folders into the webapp folder.

❑Delete the original *servlets* folder in the publishing view.

Create the WebSphere publishing stage

By default, Studio provides an empty *Production* stage. Rather than using the empty stage, we create a new tailored stage.

- ❑Create a new publishing stage. Select *Project -> Customize Publishing Stages* and enter the name *WebSphere* and click *Add*.
- ❑Copy the Test stage to the WebSphere stage. Select *Project -> Copy Publishing Stage* and copy from Test to WebSphere. This is easier than creating manually.
- ❑Switch to the WebSphere stage (*Project -> Publishing Stage -> WebSphere*).
- ❑Insert a new server named *http://localhost* (or your target host name).
- ❑Move all folders from localhost:8080 to localhost.
- ❑Delete the localhost:8080 server.
- ❑Check that the folder structure is identical to the Test stage.

This folder structure mirrors the layout of a Web application in WebSphere.

Configure the WebSphere publishing targets

We have to set up the target directories for publishing the files from Studio to WebSphere Application Server.

Select the *localhost* server and *Properties*, then click on *Define Publishing Targets*. Click on *Add* to define four new targets and set the path for each target to the proper directory:

```
topologyone servlets:  <WAS>\hosts\default_host\topologyone\servlets
topologyone web:       <WAS>\hosts\default_host\topologyone\web
topologytwo servlets:  <WAS>\hosts\secure_host\topologytwo\servlets
topologytwo web:       <WAS>\hosts\secure_host\topologytwo\web
html:                  E:\IBM HTTP Server\htdocs\U2BTop
servlet:               <WAS>\hosts\default_host\topologyone\servlets

<WAS> is d:\WebSphere\AppServer
```

We do not use the servlet target (because we have our own), but it should point to a valid directory.

Assign publishing targets to folders

In the publishing views, select the individual folders listed below and *Properties*. On the *Publish* page, select the check box *Publish this folder to a publishing target*, and select the correct target from the drop-down list:

Folder		Publishing target
U2BTop		html
web	(in webapp\topologyone)	topologyone web
web	(in webapp\topologytwo)	topologytwo web
servlets	(in webapp\topologyone)	topologyone servlets
servlets	(in webapp\topologytwo)	topologytwo servlets
servletconfig	(in webapp\topologytwo)	topologytwo servlets

Configure the Test publishing targets

We have to set up the target directories for publishing the files from Studio to the VisualAge for Java WebSphere Test Environment.

Select the *localhost:8080* server, define five new publishing targets, and set the path for each target to the proper directory:

topologyone servlets:	<WTE>\hosts\default_host\topologyone\servlets
topologyone web:	<WTE>\hosts\default_host\topologyone\web
topologytwo servlets:	<WTE>\hosts\default_host\topologytwo\servlets
topologytwo web:	<WTE>\hosts\default_host\topologytwo\web
servletconfig:	<WTE>\
html:	<WTE>\hosts\default_host\default_app\web\U2BTop
servlet:	<WTE>\hosts\default_host\topologyone\servlets

<WTE> is d:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment

Assign publishing targets to folders

In the publishing views, select the individual folders listed below and *Properties*. On the *Publish* page, select the check box *Publish this folder to a publishing target*, and select the correct target from the drop-down list:

Folder		Publishing target
U2BTop		html
web	(in webapp\topologyone)	topologyone web
web	(in webapp\topologytwo)	topologytwo web
servletconfig	(in webapp\topologytwo)	servletconfig
servlets	(in webapp\topologyone)	topologyone servlets
servlets	(in webapp\topologytwo)	topologytwo servlets

We do not want to publish the *servlets* folders because we import the Java source into the Workbench of VisualAge for Java. Select the servlets folder and remove the mark from the *Publish this folder to a publishing target*.

Interfacing with VisualAge for Java

WebSphere Studio provides for two-way communication with VisualAge for Java, and we can use VisualAge to manage changes to the Java code.

VisualAge for Java setup

We assume (and recommend) that VisualAge for Java will be your primary tool for maintaining these Java files. For successful cooperation between Studio and VisualAge for Java, check that:

- ❑VisualAge for Java is running.
- ❑The VisualAge for Java project named *ITSO Pattern Development Kit* has been defined as described in Appendix C, “Using the additional material” on page 417.
- ❑The *Remote Access to Tool API* service in VisualAge for Java has been started (*Window -> Options -> Remote Access to Tool API*).

Initial loading of files from VisualAge for Java

In Studio, you can use the *Insert -> File -> From External Source* to initially pull files into our Studio project from VisualAge. However, there is a limitation to this method. When pulling from VisualAge, you can only pull at the file level, not the package or project level. Therefore, because the PDK examples have many files, it would be extremely tedious to have to select the files individually, by class name, and insert them into the Studio package.

Therefore we loaded the files from the file system.

Updating from VisualAge for Java

Here are the steps to update Studio files for VisualAge for Java:

- ❑Select one or multiple files in a Studio folder. For Java code, always select both the source (.java) and the class files, otherwise you have to compile the source yourself afterwards.
- ❑Select *Project -> VisualAge for Java -> Update from VisualAge* (Figure 282).

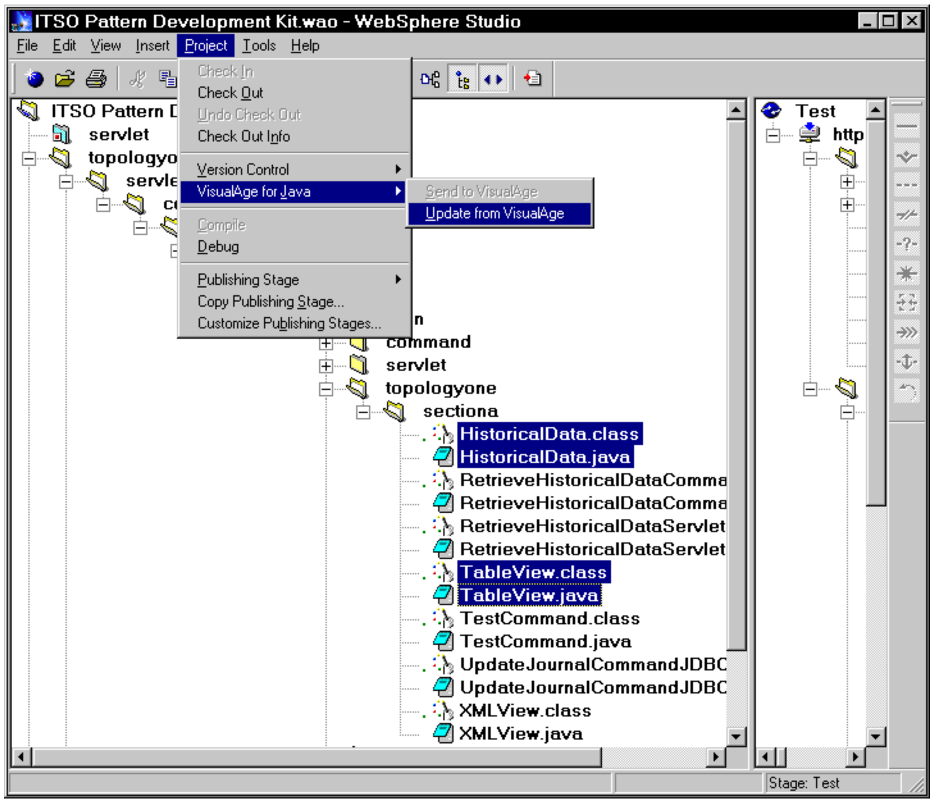


Figure 282. Updating from VisualAge for Java

You will not get a status window indicating if the update was successful or not. The new code is in checked-out status (red check mark). You can consult the check_out directory in the file system to verify that the code is there.

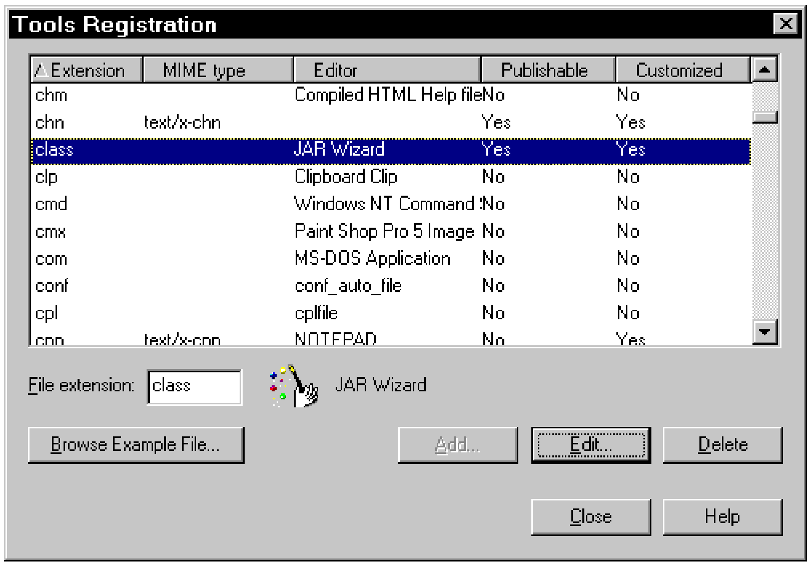
When satisfied, check-in the file to replace the master copy.

Editing Studio files with VisualAge for Java

You can choose to associate VisualAge for Java as an editor for your Java files. Here are the steps:

❑Register VisualAge for Java as a tool. Select *Tools* -> *Tools Registration* (Figure 283).

❑Find the *.class* file extension (not the .java extension).


Tools Registration

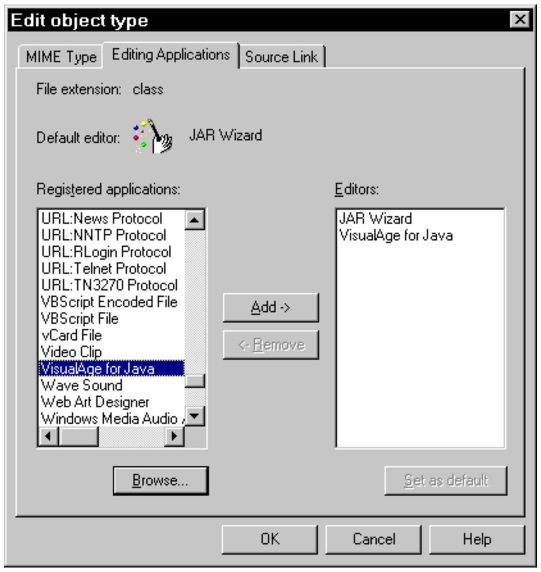
Extension	MIME type	Editor	Publishable	Customized
chm		Compiled HTML Help file	No	No
chn	text/x-chn		Yes	Yes
class		JAR Wizard	Yes	Yes
clp		Clipboard Clip	No	No
cmd		Windows NT Command	No	No
cmx		Paint Shop Pro 5 Image	No	No
com		MS-DOS Application	No	No
conf		conf_auto_file	No	No
cpl		cplfile	No	No
cpp	text/x-cpp	NOTEPAD	No	Yes

File extension: class

JAR Wizard

Figure 283. Tool registration for .class files

Click on *Edit* and the *Edit object type* dialog is displayed. Select the *Editing Applications* tab, find VisualAge for Java, and click *Add* (Figure 284).


Edit object type

File extension: class

Default editor: JAR Wizard

Registered applications:
URL:News Protocol
URL:NNTP Protocol
URL:RLogin Protocol
URL:Telnet Protocol
URL:TN3270 Protocol
VBScript Encoded File
VBScript File
vCard File
Video Clip
VisualAge for Java
Wave Sound
Web Art Designer
Windows Media Audio

Editors:
JAR Wizard
VisualAge for Java

Figure 284. Adding VisualAge for Java as an editor

Managing changes between Studio and VisualAge for Java

We recommend that you only edit your files using the VisualAge for Java editor. This way, your files should not get out of sync, as VisualAge for Java always has the most update-to-date version.

To edit a Java source file, select its class file and *Edit with -> VisualAge for Java*. (Note that you get an error prompt if you select the .java file.)

This will launch a VisualAge for Java class browser window, but it does not give focus to the window. You must switch to the class browser window yourself. You make the changes in VisualAge for Java and you save the changes, however, saving does not automatically update the files in Studio.

You have to reselect the .class and .java files, and update again from VisualAge for Java to pull the changes back into Studio.

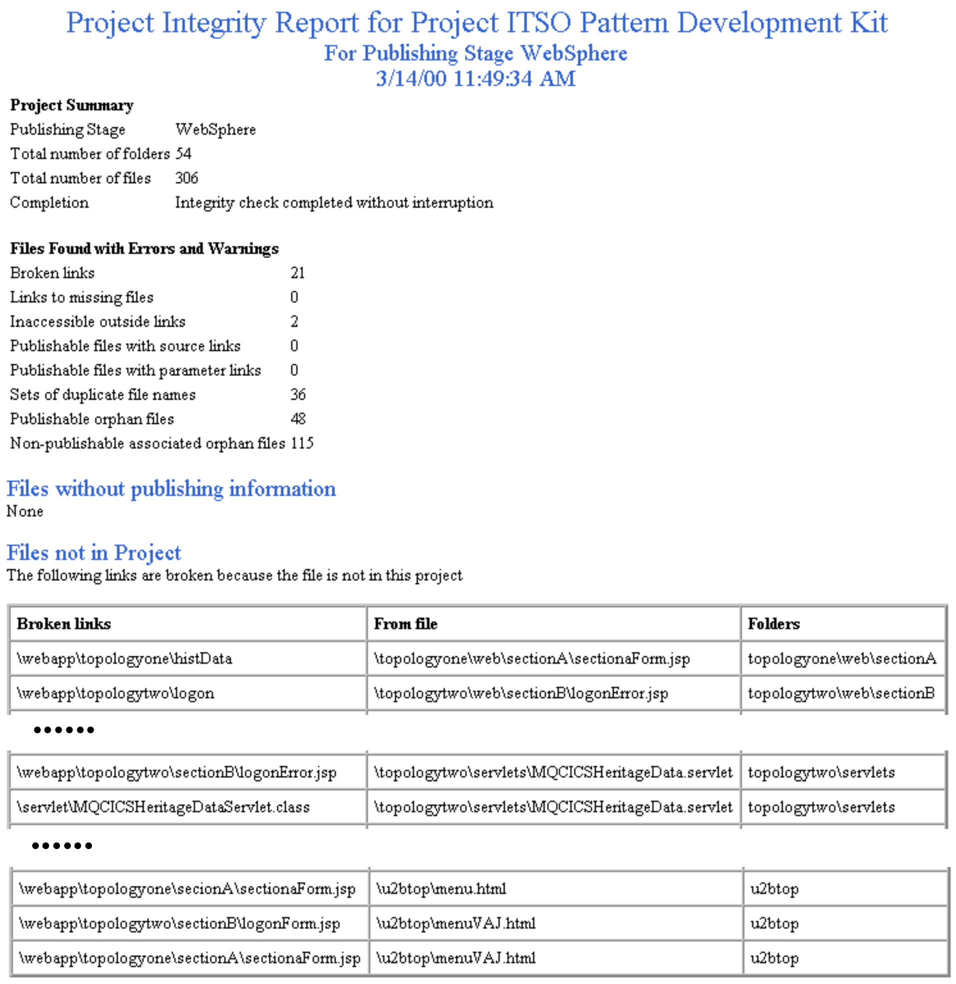
Note that the files to be edited must exist in VisualAge for Java, that is, you must have sent the files to VisualAge for Java before.

Managing the Studio project

After importing the Web site with HTML, servlets, and JSPs, we want to manage the project in WebSphere Studio.

Integrity checking for broken links

You can check the integrity of the project that was imported by running *Tools -> Check Project Integrity*. This produces a report in a browser window. The report shows many broken links (Figure 285).



Project Integrity Report for Project ITSO Pattern Development Kit
For Publishing Stage WebSphere
3/14/00 11:49:34 AM

Project Summary

Publishing Stage	WebSphere
Total number of folders	54
Total number of files	306
Completion	Integrity check completed without interruption

Files Found with Errors and Warnings

Broken links	21
Links to missing files	0
Inaccessible outside links	2
Publishable files with source links	0
Publishable files with parameter links	0
Sets of duplicate file names	36
Publishable orphan files	48
Non-publishable associated orphan files	115

Files without publishing information
None

Files not in Project
The following links are broken because the file is not in this project

Broken links	From file	Folders
\webapp\topologyone\histData	\topologyone\web\sectionA\sectionaForm.jsp	topologyone\web\sectionA
\webapp\topologytwo\logon	\topologytwo\web\sectionB\logonError.jsp	topologytwo\web\sectionB

••••••

\webapp\topologytwo\sectionB\logonError.jsp	\topologytwo\servlets\MQCICSHeritageData.servlet	topologytwo\servlets
\servlet\MQCICSHeritageDataServlet.class	\topologytwo\servlets\MQCICSHeritageData.servlet	topologytwo\servlets

••••••

\webapp\topologyone\secionA\sectionaForm.jsp	\u2btop\menu.html	u2btop
\webapp\topologytwo\sectionB\logonForm.jsp	\u2btop\menuVAJ.html	u2btop
\webapp\topologyone\sectionA\sectionaForm.jsp	\u2btop\menuVAJ.html	u2btop

Figure 285. Project integrity report

Broken links

When you import an existing site into WebSphere Studio, you get broken links if the site uses servlet aliases, or has hard-coded fully qualified references from one file to another.

The broken links in the PDK can be categorized into three categories:

- ❑Servlets invoked through a short alias name (instead of the full class name)
- ❑JSP specifications in servlet configuration files (.servlet)
- ❑JSPs called from the HTML files.

We cannot fix these broken links because of the way the application is set up.

Fixing broken links

Let construct a broken link that we can fix. Assume that the index.html file points to an image that does not exist (`splashBAD.jpg`).

In such a case, the Relations view displays a broken link. Select the broken link and *Edit Link* from the pop-up menu (Figure 286). Enter the correct name (`u2btop\images\splash5.jpg`) and click *OK* to fix the link.

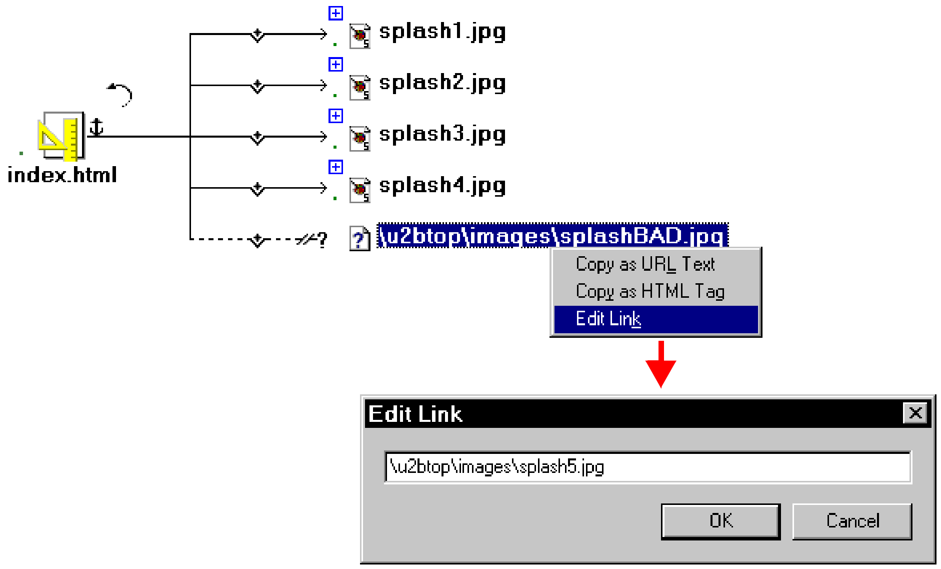


Figure 286. Fixing broken links

Publishing files

In this section, we describe how to publish the files to WebSphere and VisualAge for Java.

Publishing to WebSphere Application Server

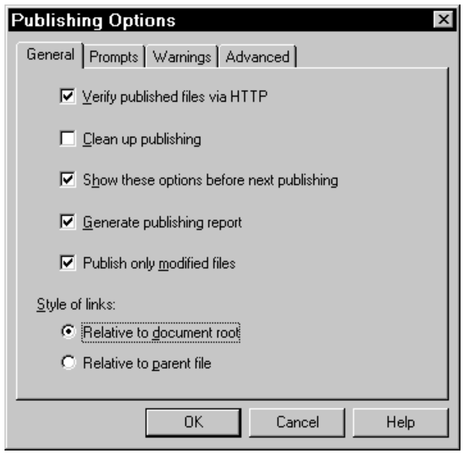
If you have successfully tested the PDK application in the WebSphere Application Server environment, then it is recommended that you test the WebSphere publishing stage.

Note. This process overwrites files in the WebSphere environment. You may want to back up the WebSphere target directory, for example:

```
d:\WebSphere\AppServer\hosts\default_host\topologyone
```

Make sure that all parts are checked-in. Select the WebSphere publishing view. Select *File* -> *Publish whole project* (or use the pop-up menu).

The Publishing Options dialog is displayed (Figure 287). Walk through all the option pages, then publish the project.



[✓] Verify published files via HTTP
[ ] Clean up publishing
[✓] Show these options before next publishing
[✓] Generate publishing report
[✓] Publish only modified files

Style of links:
[● Relative to document root]
○ Relative to parent file

Figure 287. Publishing options

You are prompted with a number of dialogs:

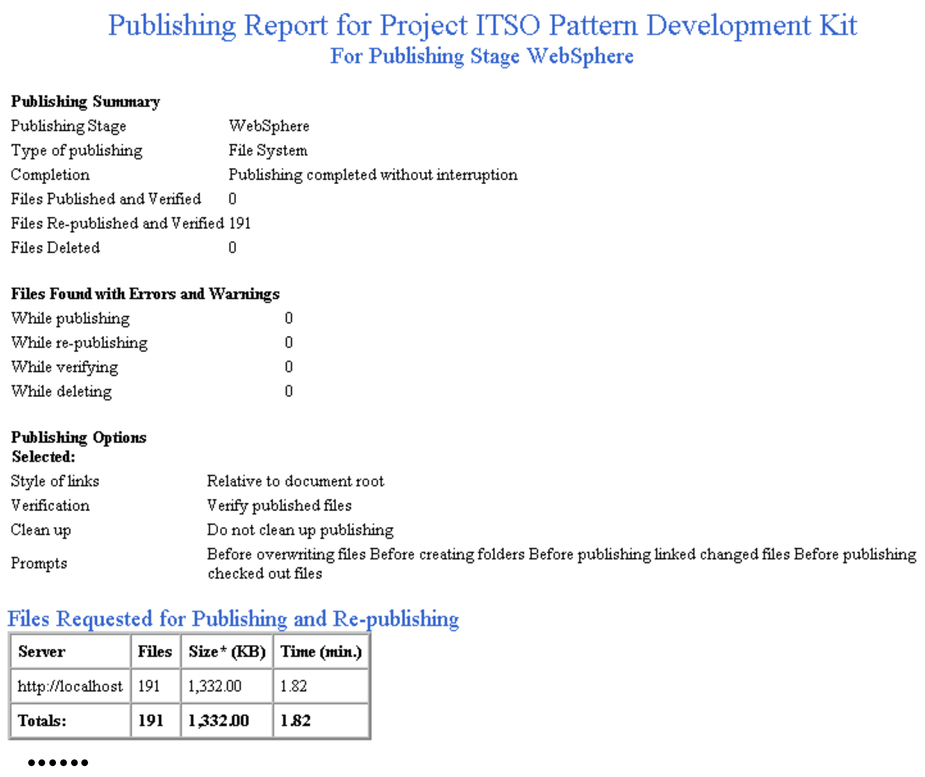
❑Files that have broken links (select all the files and continue with *OK*).

❑Folders that must be created.

❑Class files that have time stamps older than the Java source file.

Publishing report

After the publishing process completes, an HTML Publishing Report for the project and publishing stage is displayed in the browser (Figure 288).



Publishing Report for Project ITSO Pattern Development Kit
For Publishing Stage WebSphere

Publishing Summary

Publishing Stage	WebSphere
Type of publishing	File System
Completion	Publishing completed without interruption
Files Published and Verified	0
Files Re-published and Verified	191
Files Deleted	0

Files Found with Errors and Warnings

While publishing	0
While re-publishing	0
While verifying	0
While deleting	0

Publishing Options Selected:

Style of links	Relative to document root
Verification	Verify published files
Clean up	Do not clean up publishing
Prompts	Before overwriting files Before creating folders Before publishing linked changed files Before publishing checked out files

Files Requested for Publishing and Re-publishing

Server	Files	Size* (KB)	Time (min.)
http://localhost	191	1,332.00	1.82
Totals:	**191**	**1,332.00**	**1.82**

••••••

Figure 288. Publishing report

The publishing report contains details about all the published files and all the activities, such as creating directories.

Verify that the code has been placed in the correct WebSphere Application Server directories.

Publishing to VisualAge for Java

Switch the publishing view to the Test stage and repeat the publishing process. Verify that the files are placed into the correct directories of the WebSphere Test Environment.

Suppress publishing of servlet class files

For testing, you want to import the Java source into VisualAge for Java. You can suppress the publishing of the servlet class files by deleting the *servlets* folder.

You are prompted with a dialog before the action is performed. Select *Delete from current stage* (do not select *Delete from current stage and disk*).

Publishing selected folders

A better approach may be to select only individual folders for publishing, instead of the whole project. Select the *u2btop*, *webapp\topologyXXX\web*, and *webapp\topologytwo\servletconfig* folders and *Publish selected folders* from the pop-up menu.

Editing files

You can edit the HTML and JSP files using the Page Designer. However, you will notice that these files were not created with WebSphere Studio, and the Page Designer does not display the beans that are used in the JSPs.

In the JSPs, the <*jsp:usebean*> tag is placed before the <HTML> tag, with the effect that the bean is not displayed with the yellow (J) mark. If you switch to the source view and move the <jsp:usebean> tag below the <HTML> tag, you will see the (J) marker when returning to the normal view.

We suggest that you open a few JSPs and HTML files to study the normal and source code views in the Page Designer.

Appendixes

A JSP tag syntax

In this appendix we review the JSP tag syntax.

JSP tag syntax summary

See Table 19 for a summary of the all tags available in JSP 1.0.

Table 19. Summary of JSP tag syntax

Tag	Description	Syntax
Output Comment	Generates a comment that is sent to the client in the viewable page source	`<!- - comment [ <%= expression %> ] -->`
Hidden Comment	Documents the JSP page, but is not sent to the client	<%- - comment --%>
Declaration	Declares variables or methods valid in the page scripting language	`<%! declarations %>`

Tag	Description	Syntax
Expression	Contains an expression valid in the page scripting language	<%= expression %>
Scriptlet	Contains a code fragment valid in the page scripting language	<% code fragment %>
Include Directive	Includes a file of text or code in the JSP source file	<%@ include file="relativeURL" %>
Page Directive	Defines attributes that apply to an entire JSP page	<%@ page [language="java"] [extends="package.class"] [import= "{ package.class | package.*} , ..."] [session="true | false"] [buffer="none | 8kb | size kb"] [autoFlush="true | false"] [isThreadSafe="true | false"] [info="text"] [errorPage="relativeURL"] [contentType="mimeType [; charset=characterSet]" | "text/html; charset=ISO-8859-1"] [isErrorPage="true | false"] %>
Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page	<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %> custom tag: < tagPrefix:name attribute="value" + ... /> < tagPrefix:name attribute="value" + ... > other tags </ tagPrefix:name >
jsp:forward	Forwards a client request to an HTML file, JSP file or servlet for processing	<jsp:forward page="{ relativeURL | <%= expression %> }" />

jsp:getProperty	Gets the value of a Bean property so that you can display it in a JSP page	<jsp:getProperty
		name="beanInstanceName"
		property="propertyName" />

Tag	Description	Syntax
jsp:setProperty	Sets a property value or values in a bean	<jsp:setProperty name="*beanInstanceName*" { property="*" | property="*propertyName*" [param="*parameterName*"] | property="*propertyName*" value="{*string* | *<%= expression %>* }"}/>
jsp:include	Includes data in a JSP page from another file, without parsing the data	<jsp:include page="{ *relativeURL* | *<%= expression %>* }" flush="true" />
jsp:plugin	Downloads a Java plugin to the client Web browser to execute an applet or Bean	<jsp:plugin type="bean | applet" code="*classFileName*" codebase="*classFileDirName* " [name="*instanceName*"] [archive="*URIToArchive*, ... "] [align="bottom | top | middle | left | right"] [height="*displayPixels*"] [width="*displayPixels*"] [hspace="*leftRightPixels*"] [vspace="*topBottomPixels*"] [jreversion="*JREVersion* | 1.1"] [nspluginurl="*URLToPlugin*"] [iepluginurl="*URLToPlugin*"] > [<jsp:params> [<jsp:param name="*parameterName*" value="*parameterValue*" />] </jsp:params>] [<jsp:fallback> *text message for user* </jsp:fallback>] </jsp:plugin>
jsp:useBean	Locates or instantiates a Bean with a specific name and scope.	<jsp:useBean id="*beanInstanceName*" scope="page | request | session | application" { class="*package.class*" | type="*package.class* " | class="*pkg.cls*" type="*pkg.cls*" | beanName=" { *package.class* | *<%= expression %>* } " type="*package.class* "} { /> | > *other tags* </jsp:useBean> }

WebSphere specific tags

WebSphere Application Server offers a number of tags in addition to the standard tags in the JSP 1.0 specification

Table 20 describes WebSphere specific extensions to the JSP 1.0 syntax.

Table 20. IBM extensions to JSP for variable data

Tag	Description	Syntax

tsx:getProperty	The IBM extension implements all of the <jsp:getProperty> function and adds the ability to introspect a database bean that was created using the IBM extension <tsx:dbquery> or <tsx:dbmodify>.	`<tsx:getProperty name="`*`bean_name`*`"`
		`  property="`*`property_name`*`" />`

tsx:repeat	Use the <tsx:repeat> syntax to iterate over a database query results set or a repeating property in a JavaBean.	`<tsx:repeat index=`*`name`*
		`   start=`*`starting_index`*
		`   end=`*`ending_index`*`>`
		`</tsx:repeat>`

tsx:dbconnect	Use the <tsx:dbconnect> syntax to specify information needed to make a connection to a JDBC or an ODBC database. The <tsx:dbconnect> syntax does not establish the connection. Instead, the <tsx:dbquery> and <tsx:dbmodify> syntax are used to reference a <tsx:dbconnect> in the same JSP file and establish the connection.	`<tsx:dbconnect`
		`   id="`*`connection_id`*`"`
		`   userid="`*`db_user`*`"`
		`   passwd="`*`user_password`*`"`
		`   url="jdbc:`*`protocol`*`:`*`database`*`"`
		`   driver="`*`database_driver_name`*`"`
		`</tsx:dbconnect>`

Tag	Description	Syntax

tsx:userid and tsx:passwd	Instead of hardcoding the user ID and password in the <tsx:dbconnect>, you can use <tsx:userid> and <tsx:passwd> to accept user input for the values and then add that data to the request object where it can be accessed by a JSP that requests the database connection.	`<tsx:dbconnect`
		`   id="`*`connection_id`*`"`
		`   <userid>`
		`     <%= request.`
		`        getParameter("`*`userid`*`") %>`
		`   </userid>`
		`   <passwd>`
		`     <%= request.`
		`        getParameter("`*`passwd`*`") %>`
		`   </passwd>`
		`   url="jdbc:`*`protocol`*`:`*`database`*`"`
		`   driver="`*`database_driver_name`*`"`
		`</tsx:dbconnect>`

tsx:dbquery	Use the <tsx:dbquery> syntax to establish a connection to a database, submit database queries, and return the results set.	`<tsx:dbquery id="`*`query_id`*`"`
		`   connection="`*`connection_id`*`"`
		`   limit="`*`value`*`" >`
		`</tsx:dbquery>`

tsx:dbmodify	Use the <tsx:dbmodify> syntax to establish a connection to a database and then add records to a database table.	`<tsx:dbmodify`
		`   connection="`*`connection_id`*`" >`
		`</tsx:dbmodify>`

WebSphere Application Server also extends three JSP 1.0 tags by adding the "language" attribute as shown in Table 21. This enables you to use different scripting syntax for different elements of your JSP.

Table 21. WebSphere scripting language extensions (XML format only)

Syntax
`<jsp:scriptlet language="language_name">`
`<jsp:expr language="language_name">`
`<jsp:declaration language="language_name">`

B Utility servlet and utility JSP

In this appendix we list the source code and output of a utility servlet, *ServletEnvironmentSnoop*, and of a utility JSP, *WebPaths.jsp*.

These utilities can be run in WebSphere to display useful information about the current configuration, the request block, and the servlet environment.

The *ServletEnvironmentSnoop* can also run in the VisualAge for Java WebSphere Test Environment.

Utility servlet

The ServletEnvironmentSnoop utility servlet lists information about the current servlet environment and the current user request and session information.

ServletEnvironmentSnoop servlet source

Class declaration

```
package itso.servjsp.servletapi;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletEnvironmentSnoop extends HttpServlet {
```

Initialization

```
public void init(ServletConfig srvCfg) throws ServletException {
   super.init(srvCfg);
}
```

Service

```
public void service(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>ServletEnvironmentSnoop</TITLE><BODY>");
   out.println("<H2>Servlet API Example -
            ServletEnvironmentSnoop</H2><HR>");
   getReqInfo(req, out);
   getReqHeaderNames(req, out);
   getReqParmValues(req, out);
   getReqCookies(req, out);
   getReqAttributeNames(req,out);
   getInitParms(req, out);
   getHttpSessionInfo(req, out);
   getServletContextAttributes(req, out);
   out.println("</BODY></HTML>");
   out.close();
}
```

Request information

```
public void getReqInfo(HttpServletRequest req, PrintWriter out)
                            throws ServletException, IOException {
   out.println("<H4>Basic Request Information</H4>");
   out.println("This is basic information retrieved from the request
            object.<P>");
   out.println("<B>Request method:</B> " + req.getMethod() + "<BR>");
   out.println("<B>Request URI:</B> " + req.getRequestURI() + "<BR>");
   out.println("<B>Request protocol:</B> " + req.getProtocol() + "<BR>");
   out.println("<B>Request scheme:</B> " + req.getScheme() + "<BR>");
   out.println("<B>Servlet path:</B> " + req.getServletPath() + "<BR>");
   out.println("<B>Servlet name:</B> " + req.getServerName() + "<BR>");
   out.println("<B>Servlet port:</B> " + req.getServerPort() + "<BR>");
   out.println("<B>Path info:</B> " + req.getPathInfo() + "<BR>");
   out.println("<B>Path translated:</B> "+req.getPathTranslated()+"<BR>");
   out.println("<B>Character encoding:</B>"
            + req.getCharacterEncoding()+"<BR>");
   out.println("<B>Query string:</B> " + req.getQueryString() + "<BR>");
   out.println("<B>Content length:</B> " + req.getContentLength() + "<BR>");
   out.println("<B>Content type:</B> " + req.getContentType() + "<BR>");
   out.println("<B>Remote user:</B> " + req.getRemoteUser() + "<BR>");
   out.println("<B>Remote address:</B> " + req.getRemoteAddr() + "<BR>");
   out.println("<B>Remote host:</B> " + req.getRemoteHost() + "<BR>");
   out.println("<B>Authorization scheme:</B> "+req.getAuthType()+"<BR>");
   out.println("<HR>");
}
```

Request header names

```
public void getReqHeaderNames(HttpServletRequest req, PrintWriter out) {
   Enumeration e = req.getHeaderNames();
   out.println("<H4>Request Header Information</H4>");
   out.println("This is information passed in on the request header
            (http)<P>");
   if(e.hasMoreElements()) {
      while(e.hasMoreElements()) {
         String name = (String)e.nextElement();
         out.println("<B>"+name+": </B>"+req.getHeader(name)+"<BR>");
      }
   }
   else out.println("There are no request headers.");
   out.println("<HR>");
}
```

Request parameters

```
public void getReqParmValues(HttpServletRequest req, PrintWriter out) {
   out.println("<H4>Request Parameter Names/Values</H4>");
   out.println("Contains the name/value pairs of the information sent in
              on the request<P>");
   Enumeration e = req.getParameterNames();
   if (e.hasMoreElements()) {
      out.println("<H4><B>Servlet parameters
                 (Single Value style):</B></H4>"); //ex, regular fields.
      while (e.hasMoreElements()) {
         String name = (String) e.nextElement();
         out.println("<B>"+name+": </B>"+req.getParameter(name)+"<BR>");
      }
   }
   else out.println("<BR>No request parameters");
   e = req.getParameterNames();
   if (e.hasMoreElements()) {
      out.println("<H4><B>Servlet parameters
                 (Multiple Value style):</B></H1>"); //ex, checkbox's
      while (e.hasMoreElements()) {
         String name = (String) e.nextElement();
         String vals[] = (String[]) req.getParameterValues(name);
         if (vals != null) {
            out.print("<B>" + name + ": </B>");
            out.println(vals[0]);
            for (int i = 1; i < vals.length; i++)
               out.println(" " + vals[i]);
         }
         out.println("<BR>");
      }
   }
   out.println("<HR>");
}
```

Request attributes

```
public void getReqAttributeNames(HttpServletRequest req, PrintWriter out) {
   Enumeration e = req.getAttributeNames();
   out.println("<H4>Request Attribute Information</H4>");
   if(e.hasMoreElements()) {
      while(e.hasMoreElements()) {
         String name = (String)e.nextElement();
         out.println("<B>"+name+": </B>"+req.getAttribute(name)+"<BR>");
      }
   }
   else out.println("There are no request attributes");
   out.println("<HR>");
}
```

Cookies

```
public void getReqCookies(HttpServletRequest req, PrintWriter out) {
   out.println("<H4>Cookie Information</H4>");
   out.println("These are the cookies passed in on the request.
               Will be null if client cookies disabled<P>");
   Cookie[] cookies = req.getCookies();
      if(cookies != null && cookies.length > 0) {
         out.println("<H4><B>Client cookies</B></H4>");
         for(int i=0; i<cookies.length; i++) {
            out.println("<B>" + cookies[i].getName() + ": </B>" +
                        cookies[i].getValue() + "<BR>");
         }
      }
   else out.println("Cookies are null");
   out.println("<HR>");
}
```

Initialization parameters

```
public void getInitParms(HttpServletRequest req, PrintWriter out) {
   Enumeration e = getServletConfig().getInitParameterNames();
   out.println("<H4>ServletConfig Initialization Information</H4>");
   out.println("This is basic information retrieved from the ServletConfig
              file <BR>");
   out.println("(usually BigRequestHandler.servlet, if exists)<P>");
   if (e != null) {
      while (e.hasMoreElements()) {
         String param = (String) e.nextElement();
         out.println("<B>"+param+": </B> "+getInitParameter(param)+"<BR>");
      }
   }
   else out.println("ServletConfig is null");
   out.println("<HR>");
}
```

Session information

```
public void getHttpSessionInfo(HttpServletRequest req, PrintWriter out) {
   HttpSession session = req.getSession(false);
   out.println("<H4>HttpSession information</H4>");
   out.println("Will be null if session information is not utilized<P>");
   if(session != null) {
      out.println("<B>Session ID: </B>" + session.getId() + "<BR>");
      out.println("<B>Requested Session ID: </B>" +
                 req.getRequestedSessionId() + "<BR>");
      out.println("<B>Last accessed time: </B>" +
            new Date(session.getLastAccessedTime()).toString() + "<BR>");
      out.println("<B>Creation time: </B>" +
            new Date(session.getCreationTime()).toString() + "<BR>");
```

```
        String mechanism = "unknown";
        if(req.isRequestedSessionIdFromCookie()) {
           mechanism = "cookie";
        }
        else if(req.isRequestedSessionIdFromURL()) {
           mechanism = "url-encoding";
        }
        out.println("<B>Session-tracking mechanism: </B>" + mechanism +
                   "<BR>");
        String[] vals = session.getValueNames();
        if(vals != null) {
           out.println("<B>Session values: </B><BR>");
           for(int i=0; i<vals.length; i++) {
              String name = vals[i];
              out.println("<B>" + name + ": </B>" + session.getValue(name) +
                         "<BR>");
           }
        }
     }
     else out.println("Session object is null");
     out.println("<HR>");
  }
```

Context attributes

```
  public void getServletContextAttributes(HttpServletRequest req,
                                          PrintWriter out) {
     Enumeration e = getServletContext().getAttributeNames();
     out.println("<H4>ServletContext attributes</H4>");
     out.println("Contains information about the environment the servlet is
                 running under<P>");
     if(e.hasMoreElements()) {
        while(e.hasMoreElements()) {
           String name = (String)e.nextElement();
           out.println("<B>" + name + ": </B>" +
                      getServletContext().getAttribute(name) + "<BR>");
        }
     }
     out.println("<HR>");
  }
```

ServletEnvironmentSnoop servlet output

The output HTML page of the servlet is shown here without the browser frame.

Servlet API example — ServletEnvironmentSnoop

Basic Request Information

This is basic information retrieved from the request object.
Request method: GET
Request URI:
/itsoservjsp/servlet/itso.servjsp.servletapi.ServletEnvironmentSnoop
Request protocol: HTTP/1.1
Request scheme: http
Servlet path: /servlet/itso.servjsp.servletapi.ServletEnvironmentSnoop
Servlet name: 127.0.0.1
Servlet port: 80
Path info: null
Path translated: null
Character encoding: iso-8859-1
Query string: null
Content length: 0
Content type: null
Remote user: null
Remote address: 127.0.0.1
Remote host: null
Authorization scheme: null

Request Header Information

This is information passed in on the request header (http)
accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
accept-encoding: gzip, deflate
accept-language: en-us
connection: Keep-Alive
cookie: calledCount=2; sesessionid=GOL4ZAQAAAAAACIJBEE2GNA
host: 127.0.0.1
referer: http://127.0.0.1/itsoservjsp/itsoservjsp.html
user-agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)

Request Parameter Names/Values

Contains the name/value pairs of the information sent in on the request
No request parameters

Cookie Information

These are the cookies passed in on the request.
Will be null if client cookies disabled
Client cookies
calledCount: 2
sesessionid: GOL4ZAQAAAAAACIJBEE2GNA

Request Attribute Information

```
There are no request attributes
```

ServletConfig Initialization Information

```
This is basic information retrieved from the ServletConfig file
(usually BigRequestHandler.servlet, if exists)
```

HttpSession Information

```
Will be null if session information is not utilized
Session ID: GOL4ZAQAAAAAACIJBEE2GNA
Requested Session ID: GOL4ZAQAAAAAACIJBEE2GNALast accessed time:
                      Wed Mar 29 09:15:08 PST 2000
Creation time: Wed Mar 29 08:45:25 PST 2000
Session-tracking mechanism: cookie
Session values:
usersession: itso.servjsp.servletapi.SaveServletStats@22a0c0
vectorBean: itso.servjsp.jspsamples.VectorBean@3f7065
DateDisplayBean: itso.servjsp.jspsamples.DateDisplayBean@278631
```

ServletContext Attributes

```
Contains information about the environment the servlet is running under
javax.servlet.context.tempdir:
              E:\WebSphere\AppServer\temp\default_host\itsoservjsp
calledCount: 2
com.ibm.servlet.engine.webapp.WebAppServletRegistry:
              com.ibm.servlet.engine.webapp.WebAppServletRegistry@350247
com.ibm.websphere.servlet.event.ServletContextEventSource:
              com.ibm.servlet.engine.webapp.WebAppEventSource@34f729

com.ibm.websphere.servlet.application.classpath:
E:/WebSphere/AppServer/lib/ibmwebas.jar;E:/WebSphere/AppServer/properties;E
:/WebSphere/AppServer/lib/servlet.jar;E:/WebSphere/AppServer/lib/webtlsrn.j
ar;E:/WebSphere/AppServer/lib/lotusxsl.jar;E:/WebSphere/AppServer/lib/ns.ja
r;E:/WebSphere/AppServer/lib/ejs.jar;E:/WebSphere/AppServer/lib/ujc.jar;D:/
SQLLIB/java/db2java.zip;E:/WebSphere/AppServer/lib/repository.jar;E:/WebSph
ere/AppServer/lib/admin.jar;E:/WebSphere/AppServer/lib/swingall.jar;E:/WebS
phere/AppServer/lib/console.jar;E:/WebSphere/AppServer/lib/tasks.jar;E:/Web
Sphere/AppServer/lib/xml4j.jar;E:/WebSphere/AppServer/lib/x509v1.jar;E:/Web
Sphere/AppServer/lib/vaprt.jar;E:/WebSphere/AppServer/lib/iioprt.jar;E:/Web
Sphere/AppServer/lib/iioptools.jar;E:/WebSphere/AppServer/lib/dertrjrt.jar;
E:/WebSphere/AppServer/lib/sslight.jar;E:/WebSphere/AppServer/lib/ibmjndi.j
ar;E:/WebSphere/AppServer/lib/deployTool.jar;E:/WebSphere/AppServer/lib/dat
abeans.jar;E:/WebSphere/AppServer/classes;E:/JDK11~1.7/lib/classes.zip;E:/W
ebSphere/AppServer/lib/jsp10.jar;E:\WebSphere\AppServer\hosts\default_host\
itsoservjsp\servlets

com.ibm.websphere.servlet.application.host: default_host
com.ibm.websphere.servlet.application.name: itsoservjsp
```

Utility JSP

The WebPaths utility JSP only runs in the default application of WebSphere Application Server. Be sure to place the file into the Web server directory, for example:

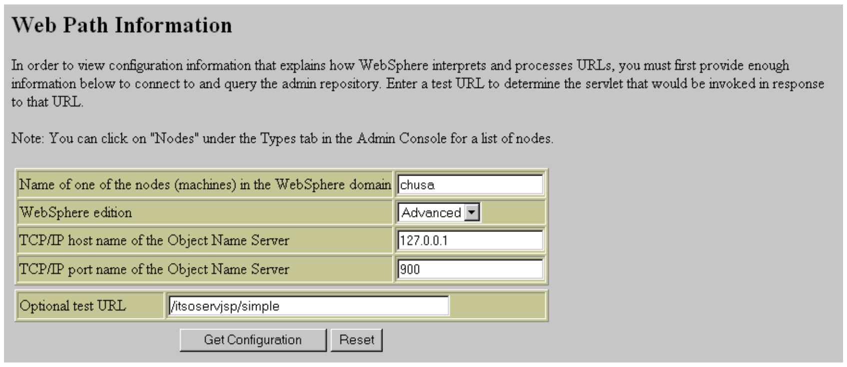
```
d:\IBM HTTP Server\htdocs
```

WebPaths.jsp source

The source code of this JSP is very long and therefore not listed here. You can find the code in `sg245755\SampCode\itsoservjsp\web\WebPaths.jsp`.

WebPaths.jsp output

The output HTML page consists of serveral parts. Some of the ouptut is shown here.



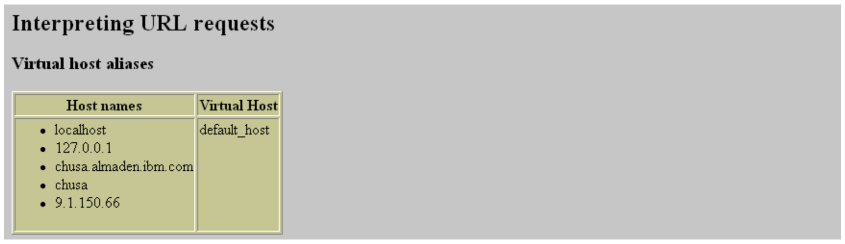
Web Path Information

In order to view configuration information that explains how WebSphere interprets and processes URLs, you must first provide enough information below to connect to and query the admin repository. Enter a test URL to determine the servlet that would be invoked in response to that URL.

Note: You can click on "Nodes" under the Types tab in the Admin Console for a list of nodes.

Name of one of the nodes (machines) in the WebSphere domain	chusa
WebSphere edition	Advanced
TCP/IP host name of the Object Name Server	127.0.0.1
TCP/IP port name of the Object Name Server	900
Optional test URL	/itsoservjsp/simple

Output if no optional test URL is entered:

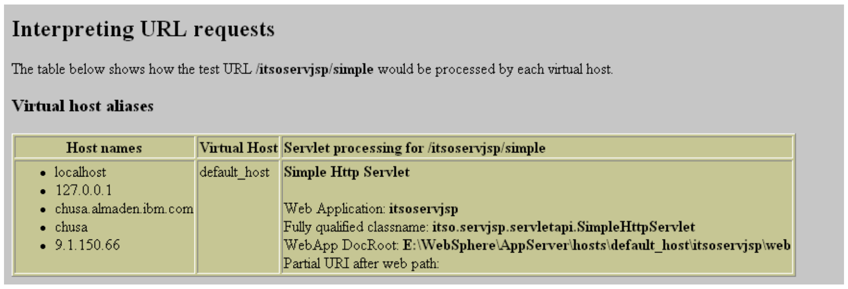


Interpreting URL requests

Virtual host aliases

Host names	Virtual Host
• localhost • 127.0.0.1 • chusa.almaden.ibm.com • chusa • 9.1.150.66	default_host

Output with an optional test URL:



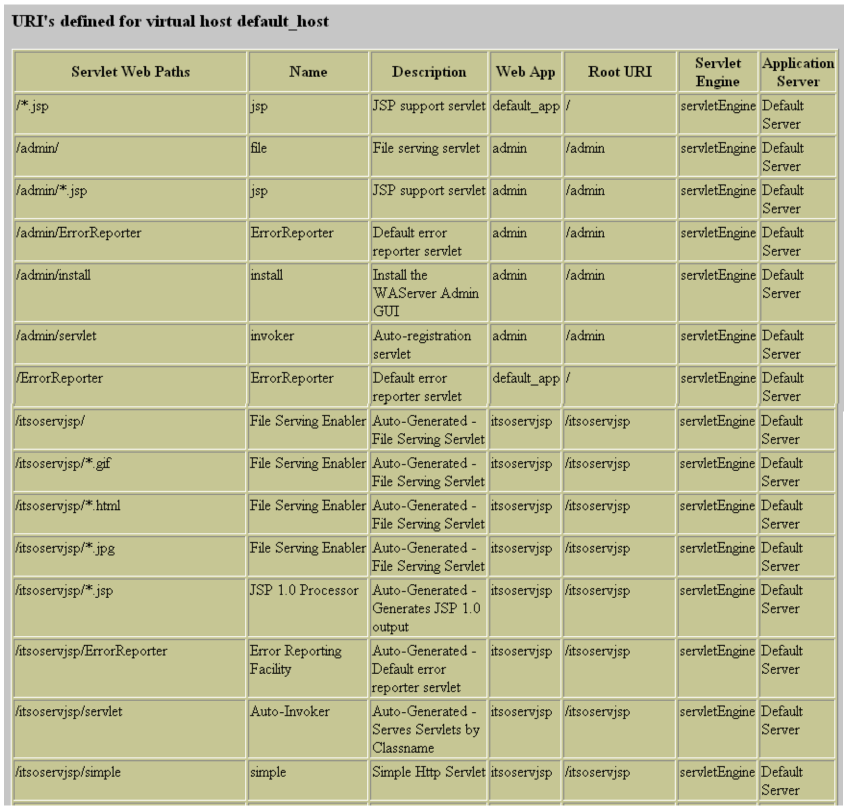
Interpreting URL requests

The table below shows how the test URL **/itsoservjsp/simple** would be processed by each virtual host.

Virtual host aliases

Host names	Virtual Host	Servlet processing for /itsoservjsp/simple
• localhost • 127.0.0.1 • chusa.almaden.ibm.com • chusa • 9.1.150.66	default_host	**Simple Http Servlet** Web Application: **itsoservjsp** Fully qualified classname: **itso.servjsp.servletapi.SimpleHttpServlet** WebApp DocRoot: **E:\WebSphere\AppServer\hosts\default_host\itsoservjsp\web** Partial URI after web path:

Output with and without optional test URL:



URI's defined for virtual host default_host

Servlet Web Paths	Name	Description	Web App	Root URI	Servlet Engine	Application Server
/*.jsp	jsp	JSP support servlet	default_app	/	servletEngine	Default Server
/admin/	file	File serving servlet	admin	/admin	servletEngine	Default Server
/admin/*.jsp	jsp	JSP support servlet	admin	/admin	servletEngine	Default Server
/admin/ErrorReporter	ErrorReporter	Default error reporter servlet	admin	/admin	servletEngine	Default Server
/admin/install	install	Install the WAServer Admin GUI	admin	/admin	servletEngine	Default Server
/admin/servlet	invoker	Auto-registration servlet	admin	/admin	servletEngine	Default Server
/ErrorReporter	ErrorReporter	Default error reporter servlet	default_app	/	servletEngine	Default Server
/itsoservjsp/	File Serving Enabler	Auto-Generated - File Serving Servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/*.gif	File Serving Enabler	Auto-Generated - File Serving Servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/*.html	File Serving Enabler	Auto-Generated - File Serving Servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/*.jpg	File Serving Enabler	Auto-Generated - File Serving Servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/*.jsp	JSP 1.0 Processor	Auto-Generated - Generates JSP 1.0 output	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/ErrorReporter	Error Reporting Facility	Auto-Generated - Default error reporter servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/servlet	Auto-Invoker	Auto-Generated - Serves Servlets by Classname	itsoservjsp	/itsoservjsp	servletEngine	Default Server
/itsoservjsp/simple	simple	Simple Http Servlet	itsoservjsp	/itsoservjsp	servletEngine	Default Server

C Using the additional material

This redbook also contains additional material on the Internet. See the appropriate section below for instructions on using or downloading each type of material.

Locating the additional material on the Internet

The CD-ROM, diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG245755/
```

Alternatively, you can go to the IBM Redbooks Web site at:

```
http://www.redbooks.ibm.com/
```

Select the *Additional materials* and open the directory that corresponds with the redbook form number.

Using the Web material

The additional Web material that accompanies this redbook includes the following:

File name	*Description*
5755samp.zip	Sample code used in Part 1
5755pdk.zip	Pattern Development Kit of Part 2
readme.txt	Description and updates

System requirements for downloading the Web material

The following system configuration is recommended for downloading the additional Web material.

Hard disk space:	10 MB minimum
Operating System:	Windows NT
Processor:	233 Mhz or better 366 MHz with WebSphere Application Server
Memory:	128 MB 256 MB with WebSphere Application Server

How to use the Web material

Create a subdirectory (folder) on your workstation and copy the contents of the Web material into this folder.

Unzip the *5755samp.zip* and *5755pdk.zip* files onto a hard drive. This creates the directory structure:

```
SG245755
   sampcode
      subdirectories for servlet and JSP samples (Part 1)
   pdk
      subdirectories for Pattern Development Kit (Part 2)
```

Pattern Development Kit

The usage of the PDK files is described in Part 2, “Pattern Development Kit: a sample application.”

See Chapter 13, “Running the PDK in WebSphere” on page 363, Chapter 14, “Running the PDK in VisualAge for Java” on page 375, and Chapter 15, “Developing the PDK using WebSphere Studio” on page 383 for detailed instructions.

Servlet and JSP sample files

The rest of this chapter describes how to use the servlet and JSP sample files.

Directory structure

The sample files are provided in subdirectories of `SG245755\sampcode` as shown in Table 22.

Table 22. Servlet and JSP sample file directory structure

Directory	Description

`cmd`	Command files that can be used to install the samples:
	`itsoenv.cmd       <=== tailor first`
	`copyVajava.cmd    <=== copy files to VA Java`
	`copyWebSphere.cmd <=== copy files to WebSphere`

`itsoservjsp`
`web`
`servlet`
	Source for servlet and JSP examples
	- HTML and JSP
	- Java and class files of the itso.servjsp.xxxxx packages

`wte`	Configuration files for the servlet engine of the VisualAge for Java WebSphere Test Environment:
	`default.servlet_engine`
	Configuration file for the *itsoservjsp* Web application:
	`itsoservjsp.webapp`

`project_` `resources`	Project resource files for VisualAge for Java WTE. A subdirectory contains the .servlet files for the ITSO Servlet JSP Redbook project.

`repository`	VisualAge for Java repository files:
	`5755samp.dat      <=== servlet and jsp samples`
	`5755pdk.dat       <=== PDK application`

`wasxml`	XML files to load definitions into WebSphere Application Server: `itsoservjsp.xml` (and others)

`studio`
 	
`itso\....`
	Archive files for WebSphereStudio projects:
	ITSO Servlet JSP Redbook, ITSO Servlet JSP Redbook Total
	Java program to load photos into EMP_PHOTO table.

If you want to use the *copyVajava.cmd* or *copyWebSphere.cmd* files provided in the *cmd* subdirectory to set up the VisualAge for Java WebSphere Test Environment or the WebSphere Application Server, you have to tailor the *itsoenv.cmd* with the correct directory names.

Test preparation

This section provides the steps necessary to configure and run the servlet and JSP examples, using either the WebSphere Application Server (as discussed in Chapter 6, “WebSphere Application Server” on page 123) or the VisualAge for Java WebSphere Test Environment (as discussed in Chapter 7, “Development and testing with VisualAge for Java” on page 167).

The sample files that are distributed must be placed into the proper directories for testing under WebSphere and VisualAge for Java.

Web application

You can either test the servlets in the default application, or you can set up a separate Web application.

We suggest that you set up a Web application called *itsoservjsp*. Some of the code assumes that such a Web application exists. Instructions for setting up the Web application are in “Creating your own Web application” on page 135 (for WebSphere) and in “WebSphere Test Environment — multiple Web applications” on page 215 (for VisualAge for Java).

WebSphere

Set up the directories manually or use the *copyWebsphere.cmd* file:

```
d:\WebSphere\AppServer\hosts\default_host\itsoservjsp\web
d:\WebSphere\AppServer\hosts\default_host\itsoservjsp\servlets
```

Copy HTML and JSP files from `sg245755\sampcode\itsoservjsp\web\` to the *web* subdirectory, and copy servlet class and configuration (.servlet) files from `sg245755\sampcode\itsoservjsp\servlets\` to the *servlets* subdirectory.

The XML files in the *wasxml* subdirectory can be used to load the *itsoservjsp* Web application into WebSphere Application Server:

```
jdbcdriver.xml                  <== load JDBC driver
datasource.xml                  <== load data source
itsoservjsp.xml                 <== load Web application
start.xml, stop.xml             <== start/stop App server
xmlImport.cmd                   <== cmd to load xml files

exportWebapp.xml                <== xml to export Web application
xmlExportWebapp.cmd             <== cmd to run the export
```

Refer to “XML configuration interface” on page 162 for instructions on how to use XML files.

VisualAge for Java

Set up the directories manually or use the *copyVajava.cmd* file:

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\
           \hosts\default_host\itsoservjsp\web
           \hosts\default_host\itsoservjsp\servlets
```

Store HTML and JSP files in the *web* subdirectory, and store utility servlet class files (SnoopServlet, ServletEngineConfigDumper) in the *servlets* subdirectory. The *itsoservjsp.webapp* configuration file must be in the servlets subdirectory as well.

Import the Java source code from `sg245755\sampcode\itsoservjsp\servlets\` into the *ITSO Servlets JSP Redbook* project. Alternatively, you can import the information from the `sg245755\sampcode\repository\5755samp.dat` file into the VisualAge for Java repository and load the project into the Workbench.

Before invoking the examples, configure the *ServletEngine* as described in "Configuring the ServletEngine class" on page 217 and start the class to bring up the WebSphere Test Envrionment.

WebSphere Studio project

You can set up a WebSphere Studio project from the archive file *ITSO Servlet JSP Redbook.war* provided in the Studio folder. This project is described in Chapter 8, "Development with WebSphere Studio" on page 227.

We also provide an archive file for a project that contains all the sample code (*ITSO Servlet JSP Redbook Total.war*).

See instructions in "Opening an archive" on page 293 on how to create a new project from an archive file and how to preserve the publishing target locations.

Servlet configuration files

Several servlets require servlet configuration files (.servlet). These files must be found in the class path by WebSphere or the WebSphere Test Environment.

WebSphere Application Server

Put the servlet configuration files into the directory

```
d:\WebSphere\AppServer\hosts\default_host\application\servlets\package
```

where *application* is either the *default_app* or the tailored application, such as *itsoservjsp*. Build the *package* subdirectories according to the full class name, for example, `itso\servjsp\servletapi`:

```
...\host\default_host\itsoservjsp\servlets\itso\servjsp\servletapi\
```

VisualAge for Java WebSphere Test Environment

The servlet configuration files (.servlet) must be in a directory that is part of the class path of the SERunner or ServletEngine class.

You can copy the files into either of these directories

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\package\
d:\IBMVJava\ide\project_resources\yourproject\package\
```

where *yourproject* is ITSO Servlet JSP Redbook, and build the *package* subdirectories according to the full class name (`itso\servjsp\servletapi`):

```
...\project_resources\ITSO Servlet JSP Redbook\itso\servjsp\servletapi\
```

You can also put the servlet configuration files under the servlets directory, but then you have to add the servlets directory to the class path of SERunner or ServletEngine:

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\
     hosts\default_host\yourapplication\servlets\
```

Note: If a servlet is invoked from an HTML file by the short alias name, then the .servlet file must be placed into a root directory of the class path and not in the package subdirectory. For example:

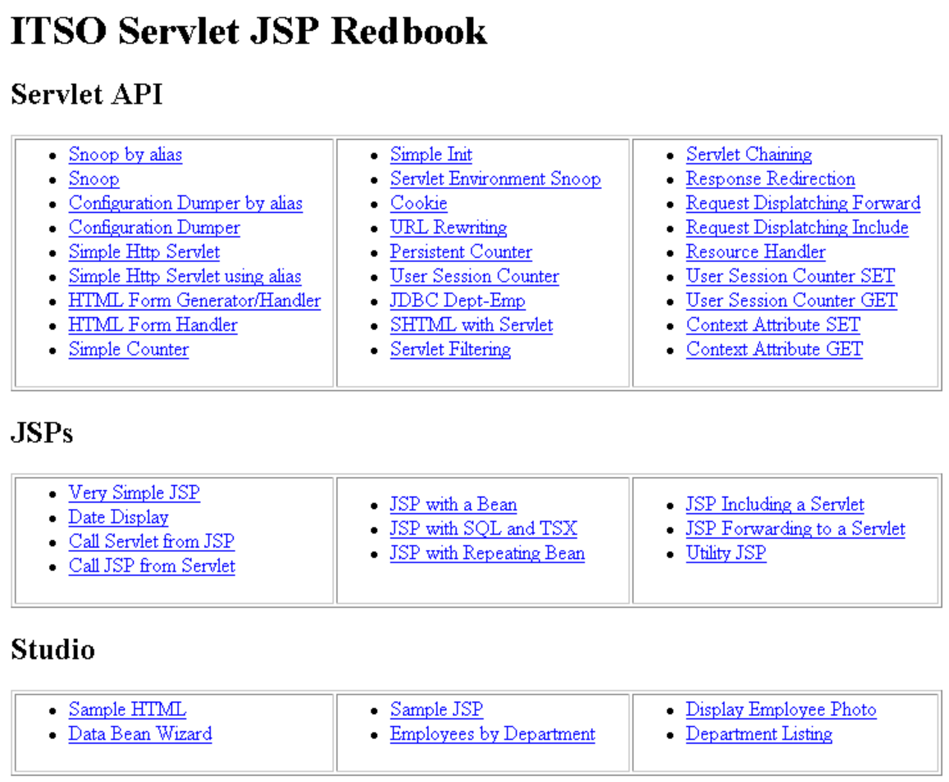
```
HTML:      <form method="post" action="/itsoservjsp/simple">

.servlet:  d:\WebSphere\AppServer\hosts\default_host\itsoservjsp\servlets
```

Testing the servlets and JSPs

We provide an HTML file named *itsoserv.html* that can be used to invoke the sample servlets and JSPs:

```
http://hostname/itsoservjsp/itsoservjsp.html         <=== WebSphere
http://hostname:8080/itsoservjsp/itsoservjsp.html    <=== VisualAge Java
```

Figure 289 shows the HTML file as it appears in a browser.



ITSO Servlet JSP Redbook

Servlet API

- Snoop by alias
- Snoop
- Configuration Dumper by alias
- Configuration Dumper
- Simple Http Servlet
- Simple Http Servlet using alias
- HTML Form Generator/Handler
- HTML Form Handler
- Simple Counter

- Simple Init
- Servlet Environment Snoop
- Cookie
- URL Rewriting
- Persistent Counter
- User Session Counter
- JDBC Dept-Emp
- SHTML with Servlet
- Servlet Filtering

- Servlet Chaining
- Response Redirection
- Request Displatching Forward
- Request Displatching Include
- Resource Handler
- User Session Counter SET
- User Session Counter GET
- Context Attribute SET
- Context Attribute GET

JSPs

- Very Simple JSP
- Date Display
- Call Servlet from JSP
- Call JSP from Servlet

- JSP with a Bean
- JSP with SQL and TSX
- JSP with Repeating Bean

- JSP Including a Servlet
- JSP Forwarding to a Servlet
- Utility JSP

Studio

- Sample HTML
- Data Bean Wizard

- Sample JSP
- Employees by Department

- Display Employee Photo
- Department Listing

Figure 289. HTML to invoke servlets and JSPs

You can also start individual examples by direct URL, such as

```
http://hostname:8080/itsoservjsp/servlet/itso.servjsp.servletapi.Xxxxxxx
```

Read the instructions on how to set up files required by individual examples.

Basic servlet examples

The following servlets are part of the basic servlet examples (see “Basic servlet examples” on page 47):

- ❑*SimpleHttpServlet* (Figure 34 on page 48): This servlet can be run with no additional environmental setup.
- ❑*HTMLFormGenerator* (Figure 39 on page 51): This servlet can be run with no additional setup.
- ❑*HTMLFormHander* (Figure 42 on page 54): Not called directly through the Servlet Launcher, it is the target of HTML page generated in *HTMLFormGeneratingServlet* above.
- ❑*SimpleCounter* (Figure 44 on page 57): This servlet can be run with no additional setup. You can reload the page multiple times from the browser to see the counter incremented.
- ❑*SimpleInitServlet* (Figure 45 on page 59): This servlet requires as input the `SimpleInitServlet.servlet` configuration (Figure 46 on page 60). See “WebSphere Studio project” on page 421 for the location of this file.
- ❑*ServletEnvironmentSnoop* (“Utility servlet” on page 408): This servlet can be run with no additional setup.

Additional servlet examples

These servlets are part of the additional examples (see “Additional servlet examples” on page 62):

- ❑*CookieServlet* (Figure 48 on page 63): This servlet requires no additional setup, other than to be invoked from a cookie-enabled browser. It requires multiple invocations to demonstrate the results. If you have two browsers installed (for example, IE and Netscape), you can demonstrate that the state is maintained by users. This is because each browser maintains its own cookies, so it is treated as two different users.
- ❑*URLServlet* (Figure 49 on page 64): This servlet requires no additional setup, but as in the cookie servlet above, requires multiple invocations to demonstrate results.
- ❑*PersistentCounter* (Figure 51 on page 66): This servlet requires a `PersistenCounter.servlet` file (Figure 50 on page 65). See “WebSphere Studio project” on page 421 for the location of this file. To adequately test this servlet, you have to start and stop the servlet runner. This servlet stores the state in a file, called `statsfile.ser`, in:

```
d:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment       <== VA Java
c:\Winnt\system32                                                      <== WebSphere
```

- ❑*UserSessionCounter* (Figure 53 on page 69): This servlet requires no additional setup, but requires multiple invocations to demonstrate results.
- ❑*JDBCInitServlet* (Figure 54 on page 70): This servlet requires a configuration file, but because it inherits from *SimpleInitServlet*, it will use the file from the superclass, unless we explicitly create one for this servlet. This servlet also assumes that DB2 has been installed, and the SAMPLE database created. The user ID and password in the `SimpleInitServlet.servlet` (or `JDBCInitServlet.servlet`) file may have to be changed to match your specific installation.
- ❑*SHTMLServlet* (Figure 57 on page 73): The .shtml extension must be associated with the JSP 0.91 compiler.

  In WebSphere, if the *jsp* support servlet is specified as 0.91, you can add the *.shtml extension to the Servlet Web Path List.

  If the jsp support servlet is for 1.0, then you have to create an additional servlet in the Web application. Name it *jsp91*, for example, with the class name *com.ibm.servlet.jsp.http.pagecompile.PageCompileServlet*, and the Web path list **.shtml* (default_host/itsoservjsp/*.shtml).

  You can define the target servlet of the <SERVLET> tag as well and call it by NAME, or you can call the target servlet by CODE (class name).

  For VisualAge for Java you have to define the JSP 0.91 support servlet (*com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet*) in the Web application (*itsoservjsp.webapp* file) and associate it with *.shtml.

```
<servlet>
   <name>jsp91</name>
   <description>JSP support servlet</description>
   <code>com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet</code>
   <init-parameter>
      <name>workingDir</name>
      <value>$server_root$/temp/default_app</value>
   </init-parameter>
   <init-parameter>
      <name>jspemEnabled</name> <value>true</value>
   </init-parameter>
   <init-parameter>
      <name>scratchdir</name>
      <value>$server_root$/temp/JSP1_0/default_app</value>
   </init-parameter>
   <init-parameter>
      <name>keepgenerated</name> <value>true</value>
   </init-parameter>
   <autostart>true</autostart>
   <servlet-path>*.shtml</servlet-path>
</servlet>
```

Servlet interaction techniques

These servlets are part of the servlet interaction techniques (see “Servlet interaction techniques” on page 73). Many of these servlets require a calling HTML page to properly invoke the results. These are included in the instructions below:

- ❑ *Servlet filtering:* The first servlet writes an output of mime-type text/Deb. This output is routed to the second servlet.

  We do not know how to tailor VisualAge for Java to make this work.

  For WebSphere you have to define the two servlets with their Web paths, for example, *FilterFirst*, `default_host/itsoservjsp/filterFirst`. Then you define the filter for the Web application on the Advanced properties page:

  ```
  Mime Type:         text/Deb
  Servlet Web Path:  FilterSecond
  ```

  Note that the servlet Web path is the name of the second servlet (FilterSecond), not the Web path name (filterSecond). We could not make it work with the Web path name.

  Test the second servlet by itself (`http://hostname/itoservjsp/filterSecond`), then run the first servlet to verify the filtering.

- ❑ *Servlet chaining:* The two servlets, ChainerFirst and ChainerSecond, have to run in sequence.

  For VisualAge for Java you have to update to the default_app.webapp file for the chainer servlet as described in “Servlet chaining” on page 212.

  For WebSphere you have to define the two servlets with their Web paths, for example, `default_host/itsoservjsp/chainFirst`. Then you define the *ChainerServlet* (in com.ibm.websphere.servlet.filter) where you define an init parameter in the Advanced properties page:

  ```
  chainer.pathlist:  /chainFirst /chainSecond
  ```

  Invoke the ChainerServlet as `http://host/itsoservjsp/chainer`.

- ❑ *Response redirection:* Requires the `HTMLFormHandlerRedirect.html` file to be in the resource path and must be invoked from a cookie-enabled browser. It requires multiple invocations to demonstrate the results. If you have two browsers installed (for example, IE and Netscape), you can demonstrate that the state is maintained by users. This is because each browser maintains its own cookies, so it is treated as two different users.

- ❑ *Request dispatching - forward:* The HTMLFormGeneratorDispatcher1 creates the HTML form that invokes the HTMLFormHandlerDispatcher1, which calls the DispatcherForward servlet.

- ❑*Request dispatching - include:* The HTMLFormGeneratorDispatcher2 creates the HTML form that invokes the HTMLFormHandlerDispatcher2, which calls the DispatcherInclude servlet.
- ❑*ResourceHandler*: This servlet requires the two HTML files ResourceHandlerHTML.html and HTMLFormHandlerRedirect.html.
- ❑*UserSessionCounterSetter* and *UserSessionCounterGetter:* These two servlets work together.
- ❑*ContextSetAttribute* and *ContextGetAttribute:* These two servlets work together.

JSP testing

To test the JSPs in VisualAge for Java, you have to import the Java source files of the servlets and beans that are used by the JSPs into the Workbench. As a minimum you have to import:

```
itso.servjsp.jspsamples                    <== import this package
itso.servjsp.servletapi.SHTMLServlet       <== import this class
```

Run individual JSPs from the HTML menu or invoke them through the browser with:

```
http://hostname/itsoservjsp/filename.jsp        <=== WebSphere
http://hostname:8080/itsoservjsp/filename.jsp   <=== VisualAge Java
```

Note that the JSPs are compiled on first usage after you start the WebSphere Test Environment, therefore the initial access is always slow.

Special instructions

The *JspSqlTsx.jsp* file must be updated with correct user ID and password to access the DB2 sample database.

D Special notices

This publication is intended to help WebSphere and VisualAge for Java developers build Web server applications using servlets, JSPs, and HTML. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server, WebSphere Studio, and VisualAge for Java Enterprise. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Application Server, WebSphere Studio, and VisualAge for Java Enterprise for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM ®	AIX
AS/400	CICS
DB2	DB2 Universal Database
MQSeries	OS/2
OS/390	S/390
SecureWay	System/390
TXSeries	VisualAge
WebSphere	Wizard

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

E Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks publications

For information on ordering these publications see “How to get IBM Redbooks” on page 437.

- ❑*Patterns for e-business: User to Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- ❑*Servlet/JSP/EJB Design and Implementation Guide,* SG24-5754 (to be published)
- ❑*The XML Files: Using XML and XSL with IBM WebSphere 3.0,* SG24-5479
- ❑*VisualAge Generator WebSphere Transactions using Generated JSPs and JavaBeans*, SG24-5636.
- ❑*WebSphere Version 3 Performance Tuning Guide,* SG24-5657
- ❑*WebSphere Application Servers: Standard and Advanced Editions,* SG24-5460
- ❑*VisualAge for Java Version 3 Persistence Builder with GUIs, Servlets, and Java Server Pages,* SG24-5426
- ❑*IBM WebSphere and VisualAge for Java Database Integration with DB2, Oracle, and SQL Server,* SG24-5471
- ❑*Developing an e-business Application for the IBM WebSphere Application Server,* SG24-5423
- ❑*The Front of IBM WebSphere, Building e-business User Interfaces,* SG24-5488
- ❑*Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429
- ❑*VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector,* SG24-5265
- ❑*Programming with VisualAge for Java Version 2,* SG24-5264, published by Prentice Hall, ISBN 0-13-021298-9, 1999 (IBM form number SR23-9016)
- ❑*VisualAge for Java Enterprise Version 2 Team Support,* SG24-5245
- ❑*Java Thin Client Systems: With VisualAge Generator - In IBM WebSphere Application Server*, SG24-5468.
- ❑*Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications*, SG24-5276
- ❑*VisualAge Java-RMI-Smalltalk, The ATM Sample from A to Z*, SG24-5418
- ❑*Using VisualAge UML Designer,* SG24-4997
- ❑*Application Development with VisualAge for Java Enterprise,* SG24-5081
- ❑*Managing Your Java Software with IBM SecureWay On-Demand Server Release 2,* SG24-5846
- ❑*Creating Java Applications with NetRexx*, SG24-2216

IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

Other resources

These publications are also relevant as further information sources:

- ❑*Java Servlet Programming*, Jason Hunter with William Crawford, published by O'Reilly, ISBN 1-56592-391-X.
- ❑*Developing JavaBeans with VisualAge for Java Version 2*, SC34-4735
- ❑*Design Patterns: Elements of Reusable Object-Oriented Software*, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, published by Addison-Wesley Professional Computing Series, ISBN 0-201-63361, 1995 (IBM form number SR28-5629)
- ❑*Managing the Software Process*, Watts S. Humphrey, published by Addison-Wesley, ISBN 0-201-18095-2.

Referenced Web sites

These Web sites are also relevant as further information sources:

- ❑`http://www.ibm.com/software/webservers/httpservers`
- ❑`http://www.ibm.com/software/webservers/appserv`
- ❑`http://www.ibm.com/software/webservers/studio`
- ❑`http://www.ibm.com/software/ad/vajava`
- ❑`http://www.ibm.com/software/data/db2/udb`
- ❑`http://www.ibm.com/software/network/directory`
- ❑`http://www.ibm.com/software/developer/web/patterns`
- ❑`http://www.ibm.com/java/jdk/download`
- ❑`http://www.alphaWorks.ibm.com/tech/bsf`
- ❑`http://www.alphaworks.ibm.com/tech/DAV4J`
- ❑`http://java.sun.com/products/servlet`
- ❑`http://java.sun.com/products/jsp`
- ❑`http://www.webdav.org`

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** `http://www.redbooks.ibm.com/`

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl`

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl`

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl`

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at `http://w3.itso.ibm.com/` and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access `MyNews` at `http://w3.ibm.com/` for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	**Order**	**Quantity**

First name | Last name

Company

Address

City | Postal code | Country

Telephone number | Telefax number | VAT number

☐ Invoice to customer number ______________________

☐ Credit card number ______________________

Credit card expiration date | Card issued to | Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

API	application programming interface
ASP	Active Server Pages
CGI	Common Gateway Interface
CICS	Customer Information Control System
DBMS	database management system
DLL	dynamic link library
E2E	end-to-end
EJB	Enterprise JavaBeans
GUI	graphical user interface
HOD	host-on-demand
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IMS	Information Management System
ITSO	International Technical Support Organization
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFC	Java Foundation Classes
JSDK	Java Servlet Development Kit
JSP	JavaServer Pages
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MVC	model-view-controller
PDK	Pattern Development Kit
RAD	rapid application development
RDBMS	relational database management system
RMI	Remote Method Invocation
SCC	software configuration control
SCM	software configuration management
SCMS	source code management systems
SQL	structured query language
SSL	secure socket layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UCM	Unified Change Management
UDB	Universal Database
UOW	unit of work
USS	UNIX System Services
URL	uniform resource locator
VCE	visual composition editor
VOB	versioned object base
WAS	WebSphere Application Server
WML	Wireless Markup Language
WTE	WebSphere Test Environment
WWW	World Wide Web
XML	eXtensible Markup Language

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5755-00 Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM

Redbooks

Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java

7/8”- (1.0” spine) -1.5”
460 <-> 788 pages

Servlet and JSP Programming

with IBM WebSphere Studio and VisualAge for Java

Teach yourself servlet and JSP programming techniques

Develop and test with WebSphere Studio and VisualAge for Java

Deploy to WebSphere Application Server

This IBM Redbook provides you with sufficient information to effectively use the WebSphere and VisualAge for Java environments to create, manage and deploy Web-based applications using methodologies centered around servlet, JavaServer Pages, and JavaBean architectures.

In Part 1 we describe the products used in our environment and provide instruction on product installation and configuration. Following this, we cover servlet and JSP programming, which provide you with both a theoretical and practical understanding of these components, together with working examples of the concepts described. For execution of the sample code, we provide information on configuring the WebSphere Application Server and deploying and running the sample Web applications in WebSphere. Using the knowledge developed in these chapters, we then provide detailed information on the development environments offered by VisualAge for Java and WebSphere Studio. These chapters assist you in using the features offered by these tools, such as integrated debugging, the WebSphere Test Environment, Studio Wizards, and publishing of Web site resources. We also describe how Rational's ClearCase product can be integrated with our environment for Software Configuration Management.

In Part 2 we describe the Pattern Development Kit sample application, including installation, configuration, and operation. We also discuss the application's use of Patterns for e-business, which presents information on some of the design decisions employed when creating the application.

This IBM Redbook is intended to be read by anyone who requires both introductory and detailed information on software development in the WebSphere environment using servlets and JavaServer Pages. We assume that you have a good understanding of Java and some knowledge of HTML.

SG24-5755-00