Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–2634

070112@15821

Contents

Preface

This guide is a starting point for developers who need information about using the various APIs and programming technologies that are supported by Sun Java^{TM} System Web Server 7.0. The guide summarizes the APIs, and provides information about configuring your server to work with server-side HTML tags and CGI programs.

Who Should Use This Book

The intended audience for this guide is the person who develops, assembles, and deploys web applications in a corporate enterprise.

This guide assumes you are familiar with the following topics:

- HTML
- Java programming language
- Structured database query languages such as SQL
- Relational database concepts
- Software development processes

Before You Read This Book

Sun Java can be installed as a stand-alone product or as a component of Sun Java Enterprise System (Java ES), a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If you are installing Sun Java as a component of Java ES, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.2`.

Sun Java Documentation Set

The Sun Java documentation set describes how to install and administer the Web Server. The URL for Sun Java documentation is `http://docs.sun.com/coll/1308.3`. For an introduction to Sun Java , refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Sun Java Documentation Set

Documentation Title	Contents
Sun Java System Web Server 7.0 Documentation Center	Web Server documentation topics organized by tasks and subject
Sun Java System Web Server 7.0 Release Notes	■ Late-breaking information about the software and documentation ■ Supported platforms and patch requirements for installing Web Server
Sun Java System Web Server 7.0 Installation and Migration Guide	Performing installation and migration tasks: ■ Installing Web Server and its various components ■ Migrating data from Sun ONE Web Server 6.0 or 6.1 to Sun Java System Web Server 7.0
Sun Java System Web Server 7.0 Administrator's Guide	Performing the following administration tasks: ■ Using the Administration and command-line interfaces ■ Configuring server preferences ■ Using server instances ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Configuring access control to secure the server ■ Using JavaPlatform Enterprise Edition (Java EE) security features ■ Deploying applications ■ Managing virtual servers ■ Defining server workload and sizing the system to meet performance needs ■ Searching the contents and attributes of server documents, and creating a text search interface ■ Configuring the server for content compression ■ Configuring the server for web publishing and content authoring using WebDAV

TABLE P–1 Books in the Sun Java Documentation Set *(Continued)*

Documentation Title	Contents
Sun Java System Web Server 7.0 Developer's Guide	Using programming technologies and APIs to do the following: ■ Extend and modify Sun Java System Web Server ■ Dynamically generate content in response to client requests and modify the content of the server
Sun Java System Web Server 7.0 NSAPI Developer's Guide	Creating custom Netscape Server Application Programmer's Interface (NSAPI) plug-ins
Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications	Implementing Java Servlets and JavaServer Pages^{TM} (JSP^{TM}) technology in Sun Java System Web Server
Sun Java System Web Server 7.0 Administrator's Configuration File Reference	Editing configuration files
Sun Java System Web Server 7.0 Performance Tuning, Sizing, and Scaling Guide	Tuning Sun Java System Web Server to optimize performance
Sun Java System Web Server 7.0 Troubleshooting Guide	Troubleshooting Web Server

Related Books

The URL for all documentation about Sun Java Enterprise System (Java ES) and its components is `http://docs.sun.com/app/docs/prod/entsys.06q4`.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value

install_dir	Represents the base installation directory for Sun Java .	Sun Java Enterprise System (Java ES) installations on the Solaris^{TM} platform:
		`/opt/SUNWwbsvr7`
		Java ES installations on the Linux and HP-UX platform:
		`/opt/sun/webserver/`
		Java ES installations on the Windows platform:
		System Drive`:\Program Files\Sun\JavaES5\WebServer7`
		Other Solaris, Linux, and HP-UX installations, non-root user:
		user's home directory`/sun/webserver7`
		Other Solaris, Linux, and HP-UX installations, root user:
		`/sun/webserver7`
		Windows, all installations:
		System Drive`:\Program Files\Sun\WebServer7`

instance_dir	Directory that contains the instance-specific subdirectories.	For Java ES installations, the default location for instances on Solaris:
		`/var/opt/SUNWwbsvr7`
		For Java ES installations, the default location for instances on Linux and HP-UX:
		`/var/opt/sun/webserver7`
		For Java ES installations, the default location for instance on Windows:
		System Drive`:\Program Files\Sun\JavaES5\WebServer7`
		For stand-alone installations, the default location for instance on Solaris, Linux, and HP-UX:
		<install_dir>
		For stand-alone installations, the default location for instance on Windows:
		System Drive`:\Program Files\sun\WebServer7`

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–3 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.

TABLE P–4 Symbol Conventions (Continued)

Symbol	Description	Example	Meaning
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Accessing Sun Resources Online

The `http://docs.sun.com` (docs.sun.com^{SM}) web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. Books are available as online files in PDF and HTML formats. Both formats are readable by assistive technologies for users with disabilities.

To access the following Sun resources, go to `http://www.sun.com`:

- Downloads of Sun products
- Services and solutions
- Support (including patches and updates)
- Training
- Research
- Communities (for example, Sun Developer Network)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "Web Server," type the following:

```
Web Server site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use "`sun.com`" in place of "`docs.sun.com`" in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is [Please define the PartNumber text entity].

◆ ◆ ◆ CHAPTER 1

Web Server Technologies Overview

This chapter provides a basic overview on various technologies that are supported in the Sun Java System Web Server 7.0.

Technologies and Enhancements in Web Server 7.0

Sun Java System Web Server 7.0 is a major new release with significant enhancements in the administration infrastructure. This is the first 64-bit version of Web Server supported on both the Solaris SPARC® and AMD64 platforms.

Web Server 7.0 provides:

- Comprehensive command-line interface support
- Consolidated configuration
- Enhanced security
- Web-based Distributed Authoring and Versioning (WEBDAV)
- Access Control List (ACL)
- URL rewriting
- Clustering support.

It also comes with a robust built-in migration tool that helps migrate applications and configurations from Web Server 6.0 and 6.1 to Sun Java System Web Server 7.0.

Sun Java System Web Server 7.0 introduces the following new features:

- Management infrastructure
- Java Web Services Developer Pack 2.0 Support
- Session replication Support
- Extensive real-time monitoring Support
- Integrated reverse proxy plug-in and FastCGI plug-in support

For more features and information, see "What's New in This Release?" in *Sun Java Enterprise System 2005Q4 Release Notes*.

Supported Standards, Protocols, and Technologies

Servlet 2.4

Java servlets are server-side Java programs that generate content in response to a client request. Servlets can be thought of as applets that run on the server side without a user interface. Servlets are invoked through URL invocation or by other servlets.

Sun Java System Web Server 7.0 supports the Java Servlet 2.4 specification.

Note – Java Servlet API version 2.4 is fully backward compatible with versions 2.1, 2.2, and 2.3. Therefore, all existing servlets continues to work without modification or recompilation.

To develop servlets, use Sun's Java Servlet API. For information about using the Java Servlet API, see the documentation provided by Sun at `http://java.sun.com/products/servlet/index.html`.

For the Java Servlet 2.4 specification, see `http://java.sun.com/products/servlet/download.html`.

For information about developing servlets in Sun Java System Web Server, see Chapter 4

JSP 2.0

Web Server 7.0 supports the JavaServer Pages (JSP) 2.0 specification. A JSP is a page, much like an HTML page, that can be viewed in a web browser. However, in addition to HTML tags, it can include a set of JSP tags and directives intermixed with Java code that extend the ability of the web page designer to incorporate dynamic content in a page. These additional features provide functionality such as displaying property values and using simple conditionals.

JSP pages can access full Java functionality by:

- Embedding Java code directly into scriptlets.
- Using server-side tags that include Java servlets.

Servlets are Java classes that must be compiled, but they can be defined and compiled by a Java programmer, who then publishes the interface to the servlet. The web page designer can access a precompiled servlet from a JSP page. For information about creating JSPs, see JavaServer Pages at `http://java.sun.com/products/jsp/index.html`

JSTL 1.1

The Java Server Pages Standard Tag Library (JSTL) encapsulates as simple tags the core functionality common to many web applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

For more information on JSTL 1.1, see `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSTL.html#wp74644`

Java Web Services Developer Pack 2.0 Support

Web Services uses a Web Services Description Language (WSDL) file to describe the service and a registry service to register and lookup the services. The Simple Object Access Protocol (SOAP) binding is the standard interoperable binding for accessing Web Services. Based on Java Web Services Developer Pack (JWSDP), Sun Java System Web Server 7.0 supports integrated Java Web Services runtime and tools, and therefore supports portable Web Services implementations. For more information, see Chapter 3.

JNDI Naming

Web Server 7.0 provides Java Naming and Directory Interface (JNDI™) support that enables web applications to look up for Java EE services such as JDBC DataSources. All functional aspects of Web Server 7.0 's JNDI implementation are essential. However, the web server's implementation does not support the `CosNaming` service required for remote objects as well as lookup of `UserTransaction`, `ORB`, JMS resources or the EJB references.

JDBC Connection Pooling

In Sun Java System Web Server 7.0, `JDBCRESOURCE` and `JDBCCONNECTIONPOOL` elements in `server.xml` have been merged into one element called `jdbc-resource` to simplify JDBC configuration. Many of the configuration parameters have been renamed. For more information on the `jdbc-resource` element, see Chapter 3, "Elements in server.xml," in *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

The simplest connection pool can be configured by using the given. In this example, the connection pool will use the ORACLE JDBC driver.

- Start `wadm`: When admin-server is running, execute the following command.

  ```
  $ ./bin/wadm --user=admin
  Please enter admin-user-password>
  Sun Java System Web Server 7.0 B01/02/2006 14:22
  wadm>
  ```
- List of configuration: Verify the list of available configuration using the following command `wadm>list-configs`.
- `jdbc-resource`: Create `jdbc-resource` configuration using the following command. For more information on all possible elements, see *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

```
wadm>create-jdbc-resource --config=test
--datasource-class=oracle.jdbc.pool.OracleDataSource jdbc/MyPool
```

- Add properties to `jdbc-resource`: Properties are primarily used to configure the driver's vendor specific properties. In the following example the properties `url`, `user`, and password is added to the `jdbc-resource`:

  ```
  wadm>create-jdbc-resource-userprop --config=test --jndi-name=jdbc/MyPool
  url=jdbc:oracle:thin:@hostname:1521:MYSID user=myuser password=mypassword
  ```

- List `jdbc-resources`: The following example displays the list of `jdbc-resources` in the configuration.

  ```
  wadm>list-jdbc-resources --config=test
  ```

- Execute the following example to list properties associated with a `jdbc-resource`.

  ```
  wadm>list-jdbc-resource-userprops --config=test --jndi-name=jdbc/MyPool
  password=mypassword user=myuser url=jdbc:oracle:thin:@hostname:1521:MYSID
  ```

- Enable the connection validation using the following example:

  ```
  wadm>set-jdbc-resource-prop --config=test --jndi-name=jdbc/MyPool
  connection-validation-table-name=test connection-validation=table
  CLI201 Command "set-jdbc-resource-prop" ran successfully
  ```

- Change the pool setting using the following example. In this example, `max-connections` are set to 100.

  ```
  wadm>set-jdbc-resource-prop --config=test
  --jndi-name=jdbc/MyPool max-connections=100
  CLI201 Command "set-jdbc-resource-prop" ran successfully
  ```

- Deploy: Deploy the configuration using the following example.

  ```
  wadm>deploy-config test
  CLI201 Command "deploy-config" ran successfully
  ```

Install JDBC driver with `jar` files in one of the following ways:

- Copy the driver's `jar` file into the server's instance library directory. `jar` files in the instance library directory will automatically load and available for server.
- Modify the JVM class-path- suffix to include JDBC driver's `jar` file. The new value will overwrite the old value of the element. For example,

  ```
  wadm> set-jvm-prop --config=test
  class-path-suffix=/export/home/lib/classes12.jar
  ```

JNDI to Access the `jdbc-resource` Within a Web Application

Using JNDI, a web application can access a JDBC connection pool by looking up the `jdbc-resource` that configures it. The `jdbc-resources` can access the name in its web descriptor. The following web descriptors example has reference to the connection pool created in the earlier example.

`WEB-INF/web.xml`

```
<web-app>
...
  <resource-ref>
    <description>JDBC Connection Pool</description>
    <res-ref-name>jdbc/myJdbc</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
...
</web-app>
```

`WEB-INF/sun-web.xml`

```
<sun-web-app>
...
        <resource-ref>
            <res-ref-name>jdbc/myJdbc</res-ref-name>
            <jndi-name>jdbc/MyPool</jndi-name>
        </resource-ref>
...
</sun-web-app>
```

In the above example, `jdbc/myJdbc` is the name by which the pool is referenced in the web application and `jdbc/MyPool` is the JNDI name of the `jdbc-resources` configured.

Code Sample: The following code is an example for using the pool in a web application.

```
Context initContext = new InitialContext();
Context webContext = (Context)initContext.lookup("java:/comp/env");

DataSource ds = (DataSource) webContext.lookup("jdbc/myJdbc");
Connection dbCon = ds.getConnection();
```

Tools Support

Sun Java Studio Enterprise 8.1

Java Studio Enterprise 8.1 lets you create Java, Web, and EJB projects from existing code.

NetBeans IDE 5.0

NetBeans IDE 5.0 is an integrated development environment to create, deploy the Java EE based web applications.

For more information, see "Using NetBeans IDE 5.0" on page 139.

Sun Java Studio Enterprise 8.1

Sun Java System Web Server 7.0 supports Sun Java Studio Enterprise 8.1, Standard Edition. You can use Sun Java Studio to assemble and deploy web applications. For more information, see "Using Sun Java Studio Enterprise 8.1" on page 138.

JSR 88 Support for Application Deployment

You can write your own JSR 88 client to deploy applications to the Sun Java System Web Server 7.0. For more information, see `http://jcp.org/en/jsr/detail?id=88`.

Lifecycle Listeners and Modules

Sun Java System Web Server 7.0 allows you to write customized classes for the various phases of the server lifecycle. For instance, a user may have a startup code that ensures that a remote data source is available for the applications. Such classes are notified of server lifecycle events. It is essential to support Java-based tasks within the Web Server environment. These tasks automatically perform actions such as starting the server and sending notification if the server shuts down. You can use this support to instantiate singletons, RMI servers, and so forth. For more information, see "List of Elements" in *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*

Session Replication

The intention of session replication is to provide sufficient session failover support. This feature covers most of usage scenarios. It is done via in-memory-backup of HTTP sessions to another server instance of the same cluster. For more information, see "Configuring Session Replication" in *Sun Java System Web Server 7.0 Administrator's Guide*

Java APIs Supported by Web Server

This section summarizes the various Java APIs supported by the Web Server, and describes how to get more information about them. For more information about non-Java APIs and programming technologies, see the *Sun Java System Web Server 7.0 Developer's Guide* and the *Sun Java System Web Server 7.0 NSAPI Developer's Guide*.

API Changes From Web Server 6.1 to Web Server 7.0

Sun Java System Web Server has the following core components:

- Web container implements the JSP 2.0, Servlet 2.4, and JSTL 1.1 specification which are part of Java EE 1.4.
- Monitoring and configuring interfaces.

CHAPTER 2

Web Applications Overview

This chapter provides a basic overview of how web applications are supported in Sun Java System Web Server 7.0 and includes the following sections:

Java Web Applications

Sun Java System Web Server 7.0 supports the Java™ Servlet 2.4 API specification and the JavaServer Pages™ (JSP™) 2.0 specification, which allows servlets and JSPs to be included in web applications.

A web application is a collection of servlets, JavaServer Pages, HTML documents, and other web resources that include image files, compressed archives, and other data. A web application can be packaged into a Web ARchive (WAR) file or exist in an open directory structure.

Web Server 7.0 also supports, SHTML and CGI, which are not Java™ Platform, Enterprise Edition (Java EE™ platform) application components. For more information about APIs and programming technologies, see *Sun Java System Web Server 7.0 Developer's Guide*.

Developing and Deploying Web Applications

This section describes how to create, secure, and deploy web applications using Sun Java System Web Server 7.0.

Creating Web Applications

This section lists the general actions to create a web application, and provides pointers to more information.

1. Create a directory for all of the web application's files. This is the web application's document root.
2. Create any needed HTML files, image files, and other static content. Place these files in the document root directory or a subdirectory where they can be accessed by other parts of the application.
3. Create any needed JSP files. For more information, see Chapter 5.
4. Create any needed servlets. For more information, see Chapter 4.
5. Compile the servlets. For details about precompiling JSPs, see "Compiling JSPs Using the Command-Line Compiler" on page 69.
6. Organize the web application as described in "Web Application Structure" on page 137.
7. Create the deployment descriptor files. For more information, see "Creating Web Deployment Descriptors" on page 140.
8. Package the web application in a `.war` (optional). For example:

   `jar -cvf` *module_name*`.war *.`
9. Deploy the web application. For more information, see "Deploying Web Applications" on page 141.

   You can create a web application manually or you can use Sun Java™ System Enterprise Studio. For more information about developing web application in Sun Java Enterprise Studio, see `http://developers.sun.com/prodtech/javatools/jscreator/learning/tutorials/2/helloweb.html`

About Securing Web Applications

You can write secure web applications for the Sun Java System Web Server with components that perform user authentication and access authorization. You can build security into web applications using the following mechanisms:

- User authentication by servlets
- User authentication for single sign-on
- User authorization by servlets
- Requesting the client certificate

For detailed information about these mechanisms, see Chapter 8.

About Deploying Applications

A web application is a collection of servlets, JavaServer Pages, HTML documents, and other web resources that might include image files, compressed archives, and other data. A web application can be packaged into a Web ARchive file (a WAR file) or exist in an open directory structure. For more information, see Chapter 9.

About Virtual Servers

A virtual server is a virtual web server that uses a unique combination of IP address, port number, and host name to identify it. You might have several virtual servers, all of which use the same IP address and port number but are distinguished by their unique host names.

When you first install Sun Java System Web Server 7.0, a default virtual server is created. You can also assign a default virtual server to each new HTTP listener you create. For details, see *Sun Java System Web Server 7.0 Administrator's Guide*.

About Default Web Applications

A web application that is deployed in a virtual server at a URL "/" becomes the default web application for that virtual server. To access the default web application for a virtual server, type the URL for the virtual server, but do not supply a context root. For example:

```
http://myvirtualserver:3184/
```

If none of the web applications under a virtual server are deployed at the URI "/", the virtual server serves HTML or JSP content from its document root, which is usually *instance_dir*`/docs`. To access this HTML or JSP content, point your browser to the URL for the virtual server, and do not supply a context root but rather specify the target file. For example:

```
http://myvirtualserver:3184/hellothere.jsp
```

Servlet Result Caching

The Sun Java System Web Server 7.0 has the ability to cache servlet or JSP results to make subsequent calls to the same servlet or JSP faster.

For more information about response caching as it pertains to servlets, see "Caching Servlet Results" on page 56.

JSP Cache Tags

JSP cache tags allow you to cache JSP page fragments within the Java engine. Each can be cached using different cache criteria. For example, if you have page fragments to view stock quotes and weather information. You can cache the stock quote fragment for 15 minutes, and the weather report fragment for 25 minutes.

For more information about JSP caching, see "JSP Cache Tags" on page 72.

Database Connection Pooling

Database connection pooling enhances the performance of servlet or JSP database interactions. For more information about Java DataBase Connectivity (JDBC™), see "Configuring JDBC Resources" in *Sun Java System Web Server 7.0 Administrator's Guide*.

Sample Applications in Web Server 7.0

Sun Java System Web Server 7.0 includes a set of sample applications, which can be found in the following directory:

install_dir/`samples`

Sample Directories

All of the sample applications are arranged in a specific and well-defined directory structure. In general, the top-level directory of a sample application includes the following:

- `src`: A directory containing all the Java source files, Deployment Descriptors, JSPs, and HTML files. Samples that use a database provide a script to populate data in the database.
- `docs`: A directory containing all documentation for the application.
- `.WAR` file: The deployable `.WAR` file for the sample application.
- `build.xml`: A file for the ANT system to build the sample application

TABLE 2–1 Sample Directories

Directory	Contents
`caching`	JSP and servlet examples that demonstrate how to cache results of JSP and servlet execution.

TABLE 2–1 Sample Directories *(Continued)*

Directory	Contents
`fastcgi`	A sample to demonstrate how to make use of the FastCGI plug-in shipped with the Web Server 7.0 (using PHP).
`i18n`	A basic Java EE web application that demonstrates how to dynamically change the display language based on user preference.
`Javamail`	A servlet that uses the Javamail API to send an email message.
`JDBC`	Java DataBase Connectivity examples are included in the following directories: ■ `blob`: A servlet that accesses BLOBs through JDBC. ■ `simple`: A basic servlet that accesses an RDBMS through JDBC. ■ `transactions`: A servlet that uses the transaction API with JDBC to control a local transaction.
`JDBC rowset`	Example showing how to use JSR-114 JDBC rowset.
`JNDI`	Java Naming and Directory Interface examples in the following directories: ■ `custom`: Demonstrates using the custom resource. ■ `external`: Demonstrates using the external resource. ■ `readenv`: Demonstrates using the environment entries specified in the `web.xml` file. ■ `url`: A servlet that uses the URL resource facility to access a resource.
`JSF 1.1`	Example showing how to use Java Server Faces (JSF) to quickly build web application user interfaces.
`JSTL`	Basic examples that demonstrate usage of the JSP Standard Tag Library.
`NSAPI plugins`	NSAPI examples.
`RMI-IIOP`	Basic example that demonstrates using a servlet to access a stateless EJB^{TM} using RMI/IIOP running in Sun Java System Web Server 7.0.

TABLE 2–1 Sample Directories *(Continued)*

Directory	Contents
`Security`	Examples demonstrating how to secure Java EE web applications via standard authentication mechanisms and access controls. Security examples are included in the following directories: ■ `basic-auth`: Demonstrates how to develop, configure, and exercise basic authentication. ■ `client-cert`: Demonstrates how to develop, configure, and exercise client certificate authentication. ■ `form-auth`: Demonstrates how to develop, configure, and exercise form-based authentication. ■ `jdbcrealm`: Demonstrates how to develop, configure, and exercise JDBC realm authentication.
Servlet and JSP example	Servlet 2.4 and JSP 2.0 examples combined into a single web application.
WS-I 1.1	Web Services interoperability 1.1 samples.
websecurity	Web Services security samples.

Building the Samples

An ANT-based build system is used to build the individual sample application. The `build.xml` file has target to compile the sources, to clean, and to build the war file. It also has targets to deploy and undeploy the application, using the corresponding CLI commands provided by the Web Server. Register application resources in the deployment target.

Documentation for the Samples

Documentation is installed along with the samples during the installation. The `index.html` in the document root of the default Web Server instance contains links to the samples documentation. In addition, you can access the documentation HTML files directly in the samples directory.

◆ ◆ ◆ CHAPTER 3

Web Services Overview

This chapter focuses on web services tasks that are performed by developers. For administrator tasks, including configuration and management information, see Appendix C, “Web Services,” in *Sun Java System Web Server 7.0 Administrator’s Guide* in Sun Java System Web Server 7.0 Administrator's Guide.

Introducing Web Services

Web Services uses a Web Services Description Language (WSDL) file to describe the service and registry service to register and look up the services. The Simple Object Access Protocol (SOAP) binding is the standard interoperable binding for accessing Web Services. There are several registry protocols available, but UDDI (Universal Description, Discovery and Integration) is probably the most recognizable. Based on Java Web Services Developer Pack, Sun Java System Web Server 7.0 supports integrated Java Web Services runtime and tools, and therefore supports portable Web Services implementations, making it interoperable with .NET clients and services using the WS-I Basic Profile. For more information on Web Services, see `http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html`.

WS-Security is an OASIS proposal for adding message-layer security to SOAP messages. It defines standardized locations and syntax by which security tokens (such as X.509 certificates and Kerberos tickets) can be carried within SOAP Headers in order to secure the contents of the SOAP message exchanges. WS-Security leverages the existing XML Digital Signature and XML Encryption specifications for capturing the results of signing and encryption operations in XML syntax. In essence, WS-Security standardizes the XML Signature and XML Encryption data blocks that are carried with a SOAP message. Sun Java System Web Server 7.0 supports the integrated WS-Security standard. In addition, this release supports JSR-196 as applicable to Web Services. Web Server provides the ability to bind SOAP-layer message-security providers and message-protection policies to the container. This binding allows the container to enforce the security on behalf of the applications.

Technologies Supported in Web Server 7.0

Web Server 7.0 supports the following technologies imported from the Java Web Services Developer Pack version 2.0. For more information, see `http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html`.

This section contains the following topics:

- "Java Web Services Developer Pack 2.0 Technologies" on page 34
- "Message Security (JSR-196)" on page 35

Java Web Services Developer Pack 2.0 Technologies

Java Web Services Developer Pack, when used with Web Server, provides an integrated development and test environment. Use this environment to quickly create sophisticated portals tailored to user roles by using web applications and web services that integrate multiple information systems. The Web Services Developer pack provides both integrated tools and middleware servers, and leverages the latest Java, XML, and Web Services standards.

- JAX-WS 2.0 (JSR 224): The Java API for XML-based JAX-WS 2.0 is the key specification for component-based web service development, and governs the standard mappings between WSDL/XML schema and Java. It is the next generation of JAX-RPC.
- JAXB 2.0: The Java API for XML Binding 2.0 (JAXB) defines an extensible mapping between the Java and XML schema type models and facilitates XML serialization of Java objects.
- JAXP 1.3.1: The Java API for XML Processing 1.3.1 (JAXP) provides a standard framework for parsing and transforming XML documents and streams using the Simple API for XML (SAX), the Document Object Model (DOM), and Extensible Stylesheets Language Transformations (XSLT).
- SOAP 1.2: SOAP is a means for encoding remote procedure calls (RPCs) and document-style information as XML.
- WSDL 1.1: WSDL is an XML-based standard for describing the external interface to a web service.
- SAAJ 1.3: SOAP with Attachments API for Java (SAAJ) defines a simple object model for creating, manipulating, and sending SOAP messages.
- XWSS: XML Web Services Security (XWSS) provides a framework within which a Web service developer can secure applications. For more information, see `http://java.sun.com/webservices/xwss/`
- Fast Infoset: The Fast Infoset specification (ITU-T Rec. X.891 | ISO/IEC 24824-1) describes an open, standards-based binary XML format that is based on the XML Information Set. It is an efficient alternative to XML. For more information, see `http://java.sun.com/webservices/docs/2.0/fastinfoset/fastinfoset1.0.1-manual.html`

Message Security (JSR-196)

In message security, security information is inserted into messages so that it travels through the networking layers and arrives with the message at the message destinations. Message security differs from transport layer security. The Message security can be used to decouple message protection from message transport so that the message remains protected after transmission. For more information on security, see
`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security2.html#wp268799`

JSR-196, as applicable to Web Services, defines a standard service provider interface by which authentication mechanism providers may be integrated with containers. Providers integrated through this interface establish the authentication identities used in container access decisions while servicing the request.

Creating Web Services

Web Services is created in different ways, depending on whether the starting point of development is a Java source file, a Java class file, or a WSDL file.

Web Services Tools

Use different tools is used to generate JAX-WS artifacts, depending on whether the starting point of Web Services development is a Java source file, a Java class file, or a WSDL file.

- `apt`: The Annotation Processing Tool (APT) tool is part of Java SE 5. It programmatically processes the annotations and generate the JAX-WS portable artifacts from an annotated Java source file.
- wsgen: The `wsgen` tool reads a service endpoint implementation class and generates all of the portable artifacts for a JAX-WS web service.
- wsimport: The `wsimport` tool reads a WSDL and generates all the required artifacts for web service development, deployment, and invocation.

Note – After using the tools, the `web.xml`, `sun-jaxws.xml`, the implementation class, and the portable JAX-WS artifacts must be bundled into a WAR that can be deployed onto a container.

▼ To Create Web Services from a Java Source

1. **Use `apt` to generate the artifacts like WSDL file, schema documents, and other Java artifacts.**
2. **Package the `web.xml`, `sun-jaxws.xml`, service endpoint interface and implementation class, value types, and generated classes, if any, into a WAR file.**

3 **Deploy the WAR to Web Server.**

▼ To Create Web Services from Java Classes

1 **Use `wsgen` to generate portable artifacts.**

2 **Package the `web.xml`, `sun-jaxws.xml`, service endpoint interface and implementation class, value types, and generated classes, if any, into a WAR file.**

3 **Deploy the WAR to Web Server.**

▼ To Create Web Services from a WSDL file

1 **Use `wsimport` to generate portable artifacts.**

2 **Implement the service endpoint.**

3 **Package the WSDL file, schema documents, `web.xml`, `sun-jaxws.xml`, service endpoint interface and implementation class, value types, and generated classes, if any, into a WAR file.**

4 **Deploy the WAR to a web container.**

Securing Web Services

Web Services Security SOAP Message Security (WS-Security) is an international standard for interoperable Web services Security that was developed in OASIS by a collaboration of all the major providers of Web services technology (including Sun Microsystems). WS-Security is a message security mechanism that uses XML Encryption and XML Digital Signature to secure Web services messages sent over SOAP. The WS-Security specification defines the use of various security tokens including X.509 certificates, SAML assertions, and username/password tokens to authenticate and encrypt SOAP Web Services messages. This section also includes:

- Understanding Message Security in the Web Server
- Securing a Web Service
- Securing a Sample Application
- Configuring the Web Server for Message Security
- Admin Console Tasks for Message Security

Understand Message Security in the Web Server

The Web Server offers integrated support for the WS-Security standard in the server-side container. This functionality is integrated with Web Services security and enforced by the container of the Web Server on behalf of applications. It can protect any web service application without requiring changes to the implementation of the application. The Web Server achieves this effect by providing facilities to bind SOAP layer message security providers and message protection policies to container and to applications deployed in container.

Assigning Message Security Roles

In the Web Server, the *system administrator* and *application deployer* roles are expected to take primary responsibility for configuring message security. In some situations, the Application Developer may also contribute, although in the typical case either of the roles may secure an existing application without changing its implementation without involving the developer. The responsibilities of the various roles are defined in the following sections.

System Administrator

The system administrator is responsible for the following tasks:

- Configuring message security providers on the Web Server.
- Managing user databases.
- Managing the keystore and truststore files.
- Deploying the samples program `fromwsdl-soap12`. The sample application is used to demonstrate the use of message layer web services security.

A system administrator uses the Admin Console to manage server security settings. Web Server stores certificates and private keys in an NSS database, the administrator can manage them using `certutil`. For an overview of message security tasks, see "Configuring the Web Server for Message Security" on page 122.

Application Deployer

The application deployer is responsible for the following tasks:

- Specifying (at application assembly) any required application-specific message protection policies if such policies have not already been specified by upstream roles (the developer or assembler).
- Modifying Sun-specific deployment descriptors to specify application-specific message protection policies information message-security-binding elements to a web service endpoint.

The application developer can turn on message security, but is not responsible for doing so. The system administrator can set the message security so that all Web Services are secured. The application deployer can set the message security when the provider or protection policy bound to the application must be different from that bound to the container.

Application Developer

The application developer or assembler is responsible for the following:

- Determining if an application-specific message protection policy is required by the application. If the policy is required, the developer or assembler works with the application deployer and ensures that the required policy is specified during application assembly.

Security Tokens and Security Mechanisms

The WS-Security specification provides an extensible mechanism for using security tokens to authenticate and encrypt SOAP Web Services messages. Use the SOAP-layer message security providers installed with the Web Server to employ username/password and X.509 certificate security tokens to authenticate and encrypt SOAP Web Services messages.

Username Tokens

The Web Server uses username tokens in the SOAP messages to establish the authentication identity of the message sender. The recipient of a message containing a Username token (within embedded password) validates that the message sender is authorized to act as the user (identified in the token) by confirming that the sender knows the user's secret password.

When using a Username token, a valid user database must be configured on the Web Server.

Digital Signatures

The Web Server uses XML Digital signatures to bind an authentication identity to the message content. Clients use digital signatures to establish their caller identity, analogous to basic authentication or SSL client certificate authentication. Digital signatures are verified by the message receiver to authenticate the source of the message content (which may be different from the sender of the message.) When using digital signatures, valid keystore and truststore files must be configured on the Web Server.

Encryption

The purpose of encryption is to modify the data such that it can only be understood by its intended audience. This modification is accomplished by substituting an encrypted element for the original content. When predicated on public key cryptography, encryption establishes the identity of the parties who can read the message.

Message Protection Policies

Message protection policies are defined for request message processing and response message processing. They are expressed in terms of requirements for source and/or recipient authentication. A source authentication policy requires that the identity of the entity that sent a message or that defined the content of a message be established in the message so that the message receiver can authenticate it. A recipient authentication policy represents a requirement that the message be sent such that the identity of the entities that can receive the message can be established by the message sender. The providers apply specific message security mechanisms so that the message protection policies are in SOAP Web Services messages.

Request and response message protection policies are defined when a provider is configured in a container. You can also configure application-specific message protection policies (at the granularity of the web service port or operation) may also be configured within the Sun-specific deployment descriptors of the application or application client. Where message protection policies are defined, the request and response message protection policies of the client must match the request and response message protection policies of the server.

Securing a Web Service

Web Services deployed on the Web Server are secured by binding SOAP layer message security providers and message protection policies to the container in which the applications are deployed or to web service endpoints served by the applications. When the Web Server is installed, SOAP layer message security providers are configured in the server-side container of the Web Server, where the container or individual applications in the container can bind to them or by individual applications in the container. During installation, the providers are configured with a simple message protection policy that, if bound to a container or to an application, would cause the source of the content in all request and response messages to be authenticated by an XML digital signature.

Use the Admin Console and CLI to perform the following tasks:

- To bind the existing providers for use by the server-side containers of the Web Server.
- To modify the message protection policies enforced by the providers.
- To create new provider configurations with alternative message protection policies.

By default, message layer security is disabled on the Web Server. To configure message layer security for the Web Server follow the steps outlined in “Configuring the Web Server for Message Security” on page 122. To use Web Services security to protect all Web Services applications applications deployed on the Web Server, follow the steps in “ To Enable Providers for Message Security” on page 42.

Once you have completed the above steps, Web Services security will be applied to all Web Services applications deployed on the Web Server.

Configuring Application-Specific Web Services Security

Configure application-specific web services during application assembly by defining message-security-binding elements in the application's Sun-specific deployment descriptors. Use these message-security-binding elements to associate a specific provider or message protection policy with a web services endpoint or service reference. You can also qualify these elements so that they apply to a specific port or method of the corresponding endpoint or referenced service.

Admin Console Tasks for Message Security

All the steps for setting up the Web Server for using message security can be accomplished using the Admin Console or the `wadm` command-line tool. For more information on message security, see *Sun Java System Web Server 7.0 Administrator's Guide*.

Support for message-layer security is integrated into the Web Server in the form of plugable authentication modules. By default, message layer security is disabled on the Web Server. The following sections provide the details for enabling, creating, editing, and deleting message security configurations and providers.

- To create a message security provider
- To enable providers for message security
- To delete a message security provider
- To enable message security for stand-alone clients

In most cases, you need to restart or reconfigure the Web Server after performing these tasks, especially to apply the change to applications already deployed on Web Server.

▼ To Create a Message Security Provider

You can add or edit or modify the message protection policy. The provider type, implementation class, and provider-specific configuration properties should be modified.

1. **In the Admin Console after you login, select the configuration you wish to modify and click Edit Configuration.**

2. **Click the Java tab.**

3. **Click the Authentication sub tab and scroll down to the SOAP Authentication.**

4. **Click New to add a provider.**

5. **Add the new provider information**

   In this page, following information is available for modification.

Note – Only `Name` and class `Name` are required. If these two fields are not specified, no authentication is applied to request or response messages. All other values are optional.

- **Name**: Identifier for this provider. Use identifier name to specify the default provider using `wadm`.
- **Class Name**: Enter the Java implementation class of the provider. Server-side providers must implement the `com.sun.enterprise.security.jauth.ServerAuthModule` interface.

  The request policy defines the authentication policy requirements associated with request processing performed by the authentication provider. Enter the policies in message-sender order. For example, a requirement that encryption occur after content means that the message receiver expects to decrypt the message before validating the signature.
- **Request Authentication Source**: Select sender, content or null to define a requirement for message-layer sender authentication (for example, username password), content authentication (for example, digital signature) or no authentication be applied to request messages. When null is specified, source authentication of the request is not required.

6 Click the Add Property button to add additional properties.

The provider shipped with the Web Server requires the below properties. If other providers are used, refer to their documentation for more information on properties and valid values.

- `server.config`: The directory and file name of an XML file that contains the server configuration information. This file is in the following location `install_dir/samples/java/webapps/webservices/security/etc /wss-server-config-2.0.xml`.

7 Click OK.

To modify provider information, click the provider and edit the values.

Equivalent wadm commands

To set the response policy, replace the word `request` in the following commands with `response`.

- Create a message security provider `msgsecurity-provider`:

```
../bin/wadm create-soap-auth-provider --port=8989 --user=admin
--password-file=/tmp/admin.passwd --config=test
--class=com.sun.xml.wss.provider.ServerSecurityAuthModule
--request-policy-auth-source=content
--request-policy-auth-recipient=before-content
--request-policy-auth-recipient=before-content
--request-policy-auth-recipient=before-content msgsecurity-provider
```

- Add the required property `server.config`:

  ```
  ../bin/wadm set-soap-auth-provider-prop --port=8989 --user=admin
  --password-file=/tmp/admin.passwd --config=test
  --provider=msgsecurity-provider request-policy-auth-source=sender
  ```
- List the provider properties:

  ```
  ../bin/wadm get-soap-auth-provider-prop --port=8989 --user=admin
  --password-file=/tmp/admin.passwd --config=test
  --provider=msgsecurity-provider
  ```

  For more information about `wadm` commands and properties, see *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*

To Enable Providers for Message Security

You can enable the message security Web Services endpoints by specifying the default provider on the server side or by specifying in the `message-binding` `element` in `sun-web.xml`.

If you enable a default provider for message security, you also need an appropriate message security on the client side.

Note – In Web Server 7.0, there is no mechanism to specify a default provider using the Admin Console. You have to specify the default provider through `wadm` CLI.

```
../bin/wadm set-config-prop --port=8989 --user=admin
--password-file=/tmp/admin.passwd --config=test
default-soap-auth-provider-name=msgsecurity-provider
```

To Delete a Message Security Provider

Follow the steps to delete a message security provider.

1. In the Admin Console, after you log in, select the configuration you wish to modify and then click Edit Configuration.
2. Click the Java tab.
3. Click the Authentication sub tab and scroll down to SOAP Authentication.
4. Click Delete.

Equivalent wadm command:

```
wadm delete-soap-auth-provider --port=8989 --user=admin
--password-file=/tmp/admin.passwd --config=test msgsecurity-provider
```

To Enable Message Security Clients

Configure the message protection policies of client so that they are equivalent to the message protection policies of the server-side providers with which they interact. A typical stand-alone client is illustrated by the bundled sample `fromwsdl-soap12`.

Deploying Web Services

Deploy a web service endpoint to the Sun Java System Web Server 7.0 just as you would deploy any servlet or application. You can use the `wadm` deploy command to publish the web service. The Sun-specific deployment descriptor file `sun-web.xml` provides optional web service enhancements using `webservice-endpoint`. For more information, see "Deploying the Server Configuration" in *Sun Java System Web Server 7.0 Administrator's Guide* and Java Web Services Developer Pack 2.0 (`http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html`).

Testing Web Services

Invoke a Web Service endpoint deployed on a web service client. A Web Service client is a Java program that makes a request to the service through a stub that acts as a proxy for the remote service. This stub is generated by the `wsimport` tool based on the WSDL file of the service.

To invoke a web service client:

1. Call `wsimport` to generate the client-side artifacts using the deployed Web Services's WSDL.

   Note – The WSDL of the deployed Web Service can be obtained from this URL `http://`*host:port*`/context-root/endpoint?wsdl`

   ---

2. Implement the client to invoke the web service.

Clients can run a deployed web service by accessing its service-endpoint-address URL, which has the following format:

`http://`*host:port*`/context-root/servlet-mapping-url-pattern`

The `context-root` is defined in the `web.xml` file. The `servlet-mapping-url-pattern` is defined in the `web.xml` file.

In the following example, the `context-root` is `my-ws` and the `servlet-mapping-url-pattern` is `/simple`. You can view the WSDL file of the deployed service in a browser by adding `?WSDL` to the end of the URL. For example:

```
http://localhost:8080/my-ws/simple?WSDL
```

Web Services Samples

The Web Server ships with a sample application named `fromwsdl-soap12`. The `fromwsdl-soap1` application features a simple web service that is implemented by Java Servlet endpoint. The service endpoint interface defines a single operation, `addNumbers`, which takes two positive integers, and returns an integer that is the sum of the two input integers.

The `fromwsdl-soap12` sample application demonstrates the WS-Security functionality to secure an existing Web Services application. The instructions helps you to enable the WS-Security functionality of the Web Server so that it is used to secure the `fromwsdl-soap12` application.

The `fromwsdl-soap12` sample application is installed in the directory: *install_dir*`/samples/java/webapps/webservices/security/fromwsdl-soap12/`.

CHAPTER 4

Developing Servlets

This chapter describes how to create servlets to control web application interactions running on a Sun Java System Web Server 7.0. In addition, this chapter describes the Sun Java System Web Server 7.0 features used to augment the Java Servlet 2.4 standards.

This chapter has the following sections:

About Servlets

Servlets, like applets, are reusable Java applications. Servlets, however, run on a web server rather than in a web browser.

Servlets provide a component-based, platform-independent method for building web-based applications, without the performance overheads, process limitations, and platform-specific liabilities of CGI programs.

Servlets supported by the Sun Java System Web Server 7.0 are based on the Java Servlet 2.4 specification.

Basic characteristics of servlets include the following:

- Servlets are created compiled and packed into a Java web application archive (.WAR) and then deployed to the Sun Java System Web Server 7.0 managed at runtime by the Sun Java System Web Server 7.0 servlet engine

- Operate on input data that is encapsulated in a request object
- Respond to a query with data encapsulated in a response object
- Provide user session information persistence between interactions
- Can be dynamically reloaded while the server is running
- Are addressable with URLs. Buttons on an application's pages often point to servlets
- Can call other servlets and/or JSPs

This section includes the following topics:

- "Servlet Data Flow" on page 46
- "Servlet Types" on page 47

Servlet Data Flow

The screen data flow scenario given below briefly explains the information flow to and from the servlet.

When a user clicks a `Submit` button, information entered in a display page is sent to a servlet. The servlet processes the incoming data and responds to generate content. Once the content is generated, the servlet creates a response page, usually by forwarding the content to a JSP. The response is sent back to the client, which sets up the next user interaction.

Information flows to and from the servlet according to the following high-level process::

▼ Data Flow From the Servlet

1. **The servlet processes the client request.**
2. **The servlet generates content.**
3. **The servlet creates a response and either:**
   a. **Sends it back directly to the client.**

      - or -

   b. **Dispatches the task to a JSP or to another servlet.**

      The servlet remains in memory, available to process another request.

Servlet Types

There are two main servlet types, generic and HTTP.

- Generic servlets:
  - Extend `javax.servlet.GenericServlet`.
  - Are protocol independent. They contain no inherent HTTP support or any other transport protocol.
- HTTP servlets:
  - Extend `javax.servlet.HttpServlet`.
  - Have built-in HTTP protocol support and are more useful in a Sun Java System Web Server 7.0 environment.

For both servlet types, you must implement the initializer method `init()` to allocate and initialize, and the destructor method `destroy()` to deallocate resources.

All servlets must implement a `service()` method, which is responsible for handling servlet requests. For generic servlets, simply override the service method to provide routines for handling requests. HTTP servlets provide a service method that automatically routes the request to another method in the servlet based on which HTTP transfer method is used. So, for HTTP servlets, override `doPost()` to process POST requests, `doGet()` to process GET requests, and so on.

Creating Servlets

To create a servlet, create a class that extends either `GenericServlet` or `HttpServlet`, overriding the appropriate methods so that it handles request.

This section discusses the following topics:

- "Creating the Class Declaration" on page 48
- "Overriding Methods" on page 48
- "Overriding init()" on page 48
- "Overriding Destroy" on page 49
- "Overriding Service, Get, and Post" on page 49
- "Accessing Parameters and Storing Data" on page 50
- "Handling Sessions and Security" on page 51
- "Handling Threading Issues" on page 51
- "Delivering Client Results" on page 53

Creating the Class Declaration

To create a servlet, write a public Java class that includes basic I/O support as well as the package `javax.servlet`. The class must extend either `GenericServlet` or `HttpServlet`. Because Sun Java System Web Server 7.0 servlets exist in an HTTP environment, the latter class is recommended. If the servlet is part of a package, you must also declare the package name so the class loader can properly locate it.

The following example header shows the HTTP servlet declaration called `MyServlet`:

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

     public class myServlet extends HttpServlet {
        ...servlet methods...
     }
```

Overriding Methods

Next, override one or more methods to provide the servlet with instructions to perform its intended task. All processing by a servlet is done on a request-by-request basis and happens in the service methods, either `service()` for generic servlets or one of the `do`*Operation*`()` methods for HTTP servlets. This method accepts request and process them according to the instructions you provide, and directs the output appropriately. You can create other methods in a servlet as well.

Overriding init()

Override the class initializer `init()` to initialize or allocate resources for the servlet instance's life, such as a counter. The `init()` method runs after the servlet is instantiated but before it accepts any requests. For more information, see the servlet API specification.

Note – When extending `GenericServlet` or `HttpServlet`, you must either override `init(ServletConfig)` to first call `super.init(ServletConfig)` or simply override the `init()` method that takes no arguments. This is done so that your servlet's `ServletConfig` will be saved locally and can later be returned by calls to your servlet's `getServletConfig()` method.

The following example of the `init()` method initializes a counter by creating a public integer variable called `thisMany`:

```
public class myServlet extends HttpServlet {
        int thisMany;
public void init (ServletConfig config) throws ServletException
        {
           super.init(config);
           thisMany = 0;
       }
       }
```

Now other servlet methods can access the variable.

Overriding Destroy

Override the class destructor `destroy()` to write log messages or to release resources that have been opened in the servlet's cycle. Resources should be appropriately closed and de-referenced so that they are recycled or garbage collected. The `destroy()` method runs just before the servlet itself is deallocated from memory. For more information, see the servlet API specification.

For example, the `destroy()` method could write a log message like the following, based on the example for "Overriding init()" on page 48 above:

```
out.println("myServlet was accessed " + thisMany " times.\\n");
```

Overriding Service, Get, and Post

When a request is made, Sun Java System Web Server 7.0 hands incoming data to the servlet engine to process request. The request includes form data, cookies, session information, and URL name-value pairs, all in a type `HttpServletRequest` object called the request object. Client metadata is encapsulated as a type `HttpServletResponse` object called the response object. The servlet engine passes both objects as the servlet's `service()` method parameters.

The default `service()` method in an HTTP servlet routes the request to another method based on the HTTP transfer method (POST, and GET). For example, HTTP POST requests are routed to the `doPost()` method, HTTP GET requests are routed to the `doGet()` method. This enables the servlet to perform different request data processing depending on the transfer method. Because the routing takes place in `service()`, there is no need to generally override `service()` in an HTTP servlet. Instead, override `doGet()`, and`doPost()` depending on the expected request type.

The automatic routing in an HTTP servlet is based simply on a call to `request.get`*Method*`()`, which provides the HTTP transfer method.

Override the `service()` method (for generic servlets) or the `doGet()` or `doPost()` methods (for HTTP servlets) to perform tasks needed to answer the request. Very often, this means collating the needed information (in the request object or in a JDBC result set object), and then passing the newly generated content to a JSP for formatting and delivery back to the client.

Most operations that involve forms use either a GET or a POST operation, so for most servlets you override either `doGet()` or `doPost()`. Implement both methods to provide for both input types or simply pass the request object to a central processing method, as shown in the following example:

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
         throws ServletException, IOException {
   doPost(request, response);
}
```

All request-by-request traffic in an HTTP servlet is handled in the appropriate `do`*Operation*`()` method, including session management, user authentication, JSPs, and accessing Sun Java System Web Server 7.0 features.

If a servlet calls the `RequestDispatcher` method `include()` or `forward()`, the request information is no longer sent as HTTP POST, and GET. In other words, if a servlet overrides `doPost()`, it may not process other servlets. if the calling servlet happens to receive its data through HTTP GET. For this reason, be sure to implement routines for all possible input types, as explained above. `RequestDispatcher` methods always call `service()`.

For more information, see “Calling a Servlet Programmatically” on page 55.

Accessing Parameters and Storing Data

Incoming data is encapsulated in a request object. For HTTP servlets, the request object type is `HttpServletRequest`. For generic servlets, the request object type is `ServletRequest`. The request object contains all request parameters, including your own request values called attributes.

To access all incoming request parameters, use the `getParameter()` method. For example:

```
String username = request.getParameter("username");
```

Set and retrieve values in a request object using `setAttribute()` and `getAttribute()`, respectively. For example:

```
request.setAttribute("favoriteDwarf", "Dwalin");
```

Handling Sessions and Security

From a Web Server's perspective, a web application is a series of unrelated server hits. There is no automatic recognition if a user has visited the site before, even if the last interaction was seconds before. A session provides a context between multiple user interactions by remembering the application state. Clients identify themselves during each interaction by a cookie, or, in the case of a cookie-less browser, by placing the session identifier in the URL.

A session object can store objects, such as tabular data, information about the application's current state, and information about the current user. Objects bound to a session are available to other components that use the same session.

For more information, see Chapter 6.

After a successful login, you should direct a servlet to establish the user's identity in a standard object called a session object that holds information about the current session, including the user's login name and any additional information to retain. Application components can then query the session object to obtain user authentication.

For more information about providing a secure user session for your application, see Chapter 8
.

Handling Threading Issues

By default, servlets are not thread-safe. The methods in a single servlet instance are usually executed numerous times simultaneously (up to the available memory limit). Each execution occurs in a different thread, though only one servlet copy exists in the servlet engine.

This is efficient system resource usage, but is dangerous because of the way Java manages memory. Because variables belonging to the servlet class are passed by reference, different threads can overwrite the same memory space as a side effect. To make a servlet (or a block within a servlet) thread-safe, do one of the following:

- Synchronize write access to all instance variables, as in public synchronized `void method()` (whole method) or `synchronized(this) {...}` (block only). Because synchronizing slows response time considerably, synchronize only blocks, or make sure that the blocks in the servlet do not need synchronization.

  Note – Web container synchronizes all access to their servlets only when `javax.servlet.SingleThreadModel` servlet is implemented.

  `javax.servlet.singleThreadModel` is deprecated in Servlet 2.4.

  For example, this servlet has a thread-safe block in `doGet()` and a thread-safe method called `mySafeMethod()`:

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

   public class myServlet extends HttpServlet {

   public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
            throws ServletException, IOException {
      //pre-processing
       synchronized (this) {
         //code in this block is thread-safe
      }
      //other processing;
      }

    public synchronized int mySafeMethod (HttpServletRequest request)
      {
      //everything that happens in this method is thread-safe
      }
   }
```

- Use the `SingleThreadModel` class to create a single-threaded servlet. When a single-threaded servlet is deployed to the Sun Java System Web Server 7.0, the servlet engine creates a servlet instance pool used for incoming requests (multiple copies of the same servlet in memory). You can change the number of servlet instances in the pool by setting the `singleThreadedServletPoolSize` property in the Sun Java System Web Server 7.0 - specific web application deployment descriptor. For more information, see Chapter 9. Servlet performance gets slower due to traffic load, and, any new requests must wait for a free instance in order to proceed. However, this is not a problem with distributed, load-balanced applications since the load automatically shifts to a less busy process.

  For example, this servlet is completely single-threaded:

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

   public class myServlet extends HttpServlet
       implements SingleThreadModel {
      servlet methods...
   }
```

Note – If a servlet is specifically written as a single thread, the servlet engine creates a pool of servlet instances to be used for incoming requests. If a request arrives when all instances are busy, it is queued until an instance becomes available. The number of pool instances is configurable in the `sun-web.xml` file, in the `singleThreadedServletPoolSize` property of the `sun-web-app` element.

Delivering Client Results

The final user interaction activity is to provide a response page to the client. The response page can be delivered in two ways, as described in the following section:

- "Creating a Servlet Response Page" on page 53
- "Creating a JSP Response Page" on page 53

Creating a Servlet Response Page

Generate the output page within a servlet by writing to the output stream. The recommended way to do this depends on the output type.

Always specify the output MIME type using `setContentType()` before any output commences, as in this example:

```
response.setContentType("text/html");
```

For textual output, such as plain HTML, create a `PrintWriter` object and then write to it using `println`. For example:

```
PrintWriter output = response.getWriter();
output.println("Hello, World\\n");
```

For binary output, write to the output stream directly by creating a `ServletOutputStream` object and then write to it using `print()`. For example:

```
ServletOutputStream output = response.getOutputStream();
output.print(binary_data);
```

Creating a JSP Response Page

Servlets can invoke JSPs in two ways, the `include()` method and the `forward()` method:

- The `include()` method in the `RequestDispatcher` interface calls a JSP and waits for it to return before continuing to process the interaction. The `include()` method can be called multiple times within a given servlet.

This example shows a JSP using `include()`:

```
RequestDispatcher dispatcher =
       getServletContext().getRequestDispatcher("JSP_URI");
 dispatcher.include(request, response);
   ... //processing continues
```

- The `forward()` method in the `RequestDispatcher` interface handles the JSP interaction control. The servlet is no longer involved with the current interactions output after invoking `forward()`, thus only one call to the `forward()` method can be made in a particular servlet.

Note – You cannot use the `forward()` method if you have already defined a `PrintWriter` or `ServletOutputStream` object.

This example shows a JSP using `forward()`:

```
RequestDispatcher dispatcher =
       getServletContext().getRequestDispatcher("JSP_URI");
 dispatcher.forward(request, response);
```

Note – Identify which JSP to call by specifying a Universal Resource Identifier (URI). The path is a `String` describing a path within the `ServletContext` scope. There is also a `getRequestDispatcher()` method in the request object that takes a `String` argument indicating a complete path.

- The `getRequestDispatcher` method is similar to the `RequestDispatcher` in the `ServletContext`. The servlet container uses the information in the request object to transform the given relative path to a complete current servlet path.

For example, `RequestDispatcher dispatcher = req.getRequestDispatcher(“foo/bar”);`

Note – In the previous example `getRequestDispatcher` does not start with “/”. In Web Server 6.1, this syntax resulted in a URI such as `/webapp/foo/bar`. In Sun Java System Web Server 7.0, the URI as `/webapp/current_servlet_path/foo/bar`

For more information about JSPs, see Chapter 5.

Invoking Servlets

You can invoke a servlet by directly addressing it from a Web page with a URL or by calling it programmatically from an already running servlet, as described in the following sections:

- "Calling a Servlet with a URL" on page 55
- "Calling a Servlet Programmatically" on page 55

Calling a Servlet with a URL

You can call servlets by using URLs embedded as links in HTML or JSP pages. The format of these URLs is as follows:

`http://`*server:port*`/`*context_root*`/servlet/`*servlet_name?name=value*

The following table describes each URL section. The left column lists the URL elements, and the right column lists descriptions of each URL element.

TABLE 4–1 URL Fields for Servlets Within a Web Application

URL Element	Description
server:port	The IP address (or host name) and optional port number. To access the default web application for a virtual server, specify only this URL section. You do not need to specify the *context_root* or *servlet_name* unless you also wish to specify name-value parameters.
context_root	The context path without the leading "/" at which the web application is installed.
`servlet`	Only needed if no `servlet-mapping` is defined in the `web.xml` file.
servlet_name	The `servlet-name` (or `servlet-mapping` if defined) as configured in the `web.xml` file.
?name=value...	Optional servlet name-value parameters.

In this example, `example` is the host name, `MortPages` is the context root, and `calcMortgage` is the servlet name:

```
http://www.example.com/MortPages/servlet/calcMortgage?rate=8.0&per=360&bal=180000
```

Calling a Servlet Programmatically

First, identify which servlet to call by specifying a URI. This is normally a path relative to the current application. For example, if your servlet is part of an application with a context root called `OfficeFrontEnd`, the URL to a servlet called `ShowSupplies` from a browser is as follows:

`http://`*server:port*`/OfficeFrontEnd/servlet/ShowSupplies?name=`*value*

You can call this servlet programmatically from another servlet in one of two ways, as described below.

- To include another servlet's output, use the `include()` method from the `RequestDispatcher` interface. This method calls a servlet by its URI and waits for it to return before continuing to process the interaction. The `include()` method can be called multiple times within a given servlet.

  For example:

  ```
  RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher("/servlet/ShowSupplies");
   dispatcher.include(request, response);
  ```

- To handle interaction control to another servlet, use the `RequestDispatcher` interface's `forward()` method with the servlet's URI as a parameter.

  This example shows a servlet using `forward()`:

  ```
  RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher("/servlet/ShowSupplies");
   dispatcher.forward(request, response);
  ```

Servlet Output

By default, the `System.out` and `System.err` output of servlets is sent to the server log. During startup, server log messages are echoed to the `System.err` output. Also by default, on Windows, there is no console created for the `System.err` output.

You can change these defaults using the Admin Console. For more information, see “Setting Up Logging for Your Server” in *Sun Java System Web Server 7.0 Administrator’s Guide*.

Caching Servlet Results

Sun Java System Web Server 7.0 can cache the results of invoking a servlet, a JSP, or any URL pattern to make subsequent invocations of the same servlet, JSP, or URL pattern faster. The Sun Java System Web Server 7.0 caches the request results for a specific amount of time. In this way, if another data call occurs, the Sun Java System Web Server 7.0 can return the cached data instead of performing the operation again. For example, if your servlet returns a stock quote that updates every 5 minutes, you set the cache to expire after 300 seconds.

If caching is enabled, Sun Java System Web Server 7.0 has the ability to cache the results produced by the servlet `javax.servlet.RequestDispatcher.include()` or `javax.servlet.RequestDispatcher.forward()`. In Sun ONE releases, the result generated by a resource for which caching was enabled was never cached if that resource was the target of a `javax.servlet.RequestDispatcher.include()` or `javax.servlet.RequestDispatcher.forward()`. It was cached only if the resource was target of the initial request.

```
<sun-web-app>
<cache enable="true">
        <cache-mapping>
                <servlet-name>TestServlet</servlet-name>
                <dispatcher>REQUEST</dispatcher>
                <dispatcher>FORWARD</dispatcher>
        </cahce-mapping>
        </cache>
      </sun-web-app>
```

Whether to cache results and how to cache them depends on the data involved. For example, you no need to cache the results of a quiz submission, because the input to the servlet is different each time. However, you could cache a high-level report showing demographic data taken from quiz results that is updated once an hour.

You can define how a Sun Java System Web Server 7.0 web application handles response caching by editing specific fields in the `sun-web.xml` file. In this way, you can create programmatically standard servlets that still take advantage of this valuable Sun Java System Web Server 7.0 feature.

For more information about JSP caching, see "JSP Cache Tags" on page 72.

Caching Features

Sun Java System Web Server 7.0 has the following web application response caching capabilities:

- Caching is configurable based on the servlet name or the URI.
- When caching is based on the URI, the URI can include user-specified parameters in the query string. For example, a response from `/garden/catalog?category=roses` is different from a response from `/garden/catalog?category=lilies`. These responses are stored under different keys in the cache.
- Cache size, entry timeout, and other caching behaviors are configurable.
- Entry timeout is measured from the time an entry is created or refreshed. You can override this timeout for an individual cache mapping by specifying the `cache-mapping` subelement timeout.

- You can determine caching criteria programmatically by writing a cache helper class. For example, if a servlet only knows when a back-end data source was last modified, you can write a helper class to retrieve the last-modified timestamp from the data source and decide whether to cache the response based on that timestamp. See "CacheHelper Interface" on page 58.
- You can determine cache key generation programmatically by writing a cache key generator class. See "CacheKeyGenerator Interface" on page 61.
- All non-ASCII request parameter values specified in cache key elements must be URL encoded. The caching subsystem attempts to match the raw parameter values in the request query string.
- The newly updated classes impact what gets cached, the web container clears the cache during dynamic deployment or reloading of classes.
- The following `HttpServletRequest` request attributes are exposed:
  - `com.sun.appserv.web.cachedServletName`, the cached servlet target
  - `com.sun.appserv.web.cachedURLPattern`, the URL pattern being cached

Default Cache Configuration

If you enable caching but do not provide any special configuration for a servlet or JSP, the default cache configuration is as follows:

- The default cache timeout is 30 seconds.
- Only the HTTP GET method is eligible for caching.
- HTTP requests with cookies or sessions automatically disable caching.
- No special consideration is given to `Pragma:`, `Cache-control:`, or `Vary:` headers.
- The default key consists of the servlet path (minus `pathInfo` and the query string).
- A least recently-used list is maintained to delete cache entries if the maximum cache size is exceeded.
- Key generation concatenates the servlet path with key field values, if any are specified.

CacheHelper Interface

Here is the `CacheHelper` interface:

```
package com.sun.appserv.web.cache;
import java.util.Map
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
/** CacheHelper interface is an user-extensible interface to customize:
 * a) the key generation b) whether to cache the response.
 */
public interface CacheHelper {
// name of request attributes
   public static final String ATTR_CACHE_MAPPED_SERVLET_NAME =
                            "com.sun.appserv.web.cachedServletName";
   public static final String ATTR_CACHE_MAPPED_URL_PATTERN =
                            "com.sun.appserv.web.cachedURLPattern";
public static final int TIMEOUT_VALUE_NOT_SET = -2;
/** initialize the helper
    * @param context the web application context this helper belongs to
    * @exception Exception if a startup error occurs
    */
   public void init(ServletContext context, Map props) throws Exception;
/** getCacheKey: generate the key to be used to cache this request
    * @param request incoming <code>HttpServletRequest</code> object
    * @returns the generated key for this requested cacheable resource.
    */
   public String getCacheKey(HttpServletRequest request);
/** isCacheable: is the response to given request cacheable?
    * @param request incoming <code>HttpServletRequest</code> object
    * @returns <code>true</code> if the response could be cached. or
    * <code>false</code> if the results of this request must not be cached.
    */
   public boolean isCacheable(HttpServletRequest request);
/** isRefreshNeeded: is the response to given request to be refreshed?
    * @param request incoming <code>HttpServletRequest</code> object
    * @returns <code>true</code> if the response needs to be refreshed.
    * or return <code>false</code> if the results of this request
    * dont need to be refreshed.
    */
   public boolean isRefreshNeeded(HttpServletRequest request);
/** get timeout for the cached response.
    * @param request incoming <code>HttpServletRequest</code> object
    * @returns the timeout in seconds for the cached response; a return
    * value of -1 means the response never expires and a value of -2 indicates
    * helper cannot determine the timeout (container assigns default timeout)
   */
public int getTimeout(HttpServletRequest request);
/**
    * Stop the helper from active use
    * @exception Exception if an error occurs
    */
   public void destroy() throws Exception;
}
```

Caching Example

Here is an example cache element in the sun-web.xml file:

```
<cache max-capacity="8192" timeout="60">
   <cache-helper name="myHelper" class-name="MyCacheHelper"/>
   <cache-mapping>
      <servlet-name>myservlet</servlet name>
      <timeout name="timefield">120</timeout>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </cache-mapping>
   <cache-mapping>
      <url-pattern> /catalog/* </url-pattern>
      <!-- cache the best selling category; cache the responses to
        -- this resource only when the given parameters exist. Cache
        -- only when the catalog parameter has qliliesq or qrosesq
        -- but no other catalog varieties:
        -- /orchard/catalogbest&category=lilies
        -- /orchard/catalogbest&category=roses
        -- but not the result of
        -- /orchard/catalog?best&category=wild
      -->
      <constraint-field name=best scope=request.parameter/>
      <constraint-field name=category scope=request.parameter>
        <value> roses </value>
        <value> lilies </value>
      </constraint-field>
      <!-- Specify that a particular field is of given range but the
        -- field does not need to be present in all the requests -->
      <constraint-field name=SKUnum scope=request.parameter>
        <value match-expr=qin-range> 1000 - 2000 </value>
      </constraint-field>
      <!-- cache when the category matches with any value other than
        -- a specific value -->
      <constraint-field name="category" scope="request.parameter>
        <value match-expr="equals" cache-on-match-failure="true">bogus</value>
      </constraint-field>
   </cache-mapping>
   <cache-mapping>
      <servlet-name> InfoServlet </servlet name>
      <cache-helper-ref>myHelper</cache-helper-ref>
   </cache-mapping>
</cache>
```

For more information about the sun-web.xml caching settings, see "Caching Elements" on page 183.

CacheKeyGenerator Interface

The built-in default CacheHelper implementation allows web applications to customize the key generation. An application component (in a servlet or JSP) can set up a custom CacheKeyGenerator implementation as an attribute in the ServletContext.

The name of the context attribute is configurable as the value of the cacheKeyGeneratorAttrName property in the default-helper element of the sun-web.xml deployment descriptor. For more information, see "default-helper" on page 187.

Here is the CacheKeyGenerator interface:

```
package com.sun.appserv.web.cache;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
/** CacheKeyGenerator: a helper interface to generate the key that
 * is used to cache this request.
 *
 * Name of the ServletContext attribute implementing the
 * CacheKeyGenerator is configurable via a property of the
 * default-helper in sun-web.xml:
 * <default-helper>
 * <property
 * name="cacheKeyGeneratorAttrName"
 * value="com.acme.web.MyCacheKeyGenerator" />
 * </default-helper>
 *
 * Caching engine looks up the specified attribute in the servlet
 * context; the result of the lookup must be an implementation of the
 * CacheKeyGenerator interface.
 */
public interface CacheKeyGenerator {
/** getCacheKey: generate the key to be used to cache the
    * response.
    * @param context the web application context
    * @param request incoming <code>HttpServletRequest</code>
    * @returns key string used to access the cache entry.
    * if the return value is null, a default key is used.
    */
   public String getCacheKey(ServletContext context,
                           HttpServletRequest request);
}
```

Maximizing Servlet Performance

Consider the following guidelines for improving servlet performance:

- Increase the Java heap size to help garbage collection. The Java heap size can be adjusted by adjusting the values specified to the —Xms and —Xmx `jvm- options` in `server.xml`. For example, `<jvm-options>-Xms128m-Xmx256m</jvm-options>` sets the minimum Java heap size to 128 MB and 256 MB. For more information, see *Sun Java System Web Server 7.0 Administrator's Guide*.
- Set the stack space if applications use deep recursion, or in any cases where very complex JSP pages are used.

  You can set the stack space using `set-thread-pool-prop` wadm command. For example, `$wadm set-thread-pool-prop [other args] stack-size=544288.` For more information, see *Sun Java System Web Server 7.0 NSAPI Developer's Guide*.
- Use the NSAPI cache improve servlet performance in cases where the `obj.conf` configuration file has many directives. To enable the NSAPI cache, include the following line in `magnus.conf`:

  ```
  Init fn="nsapi-cache-init" enable=true
  ```
- Check the session ID generator which is used for servlet sessions employs cryptographically unique random number generation algorithms. This may present a performance problem on older, slow machines. For more information, see Chapter 6.
- Reduce the number of directories in the classpath.
- Disable dynamic reloading.
- Disable the Java Security Manager.

Servlet Internationalization Issues

This section describes how the Sun Java System Web Server 7.0 determines the character encoding for the servlet request and the servlet response. For encodings, see: `http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html`

Servlet Request

When processing the servlet request, the server uses the following order of precedence, first to last, to determine the character encoding for the request parameters:

- The `ServletRequest.setCharacterEncoding()` method.
- A hidden field in the form, if specified using the `form-hint-field` attribute of the `parameter-encoding` element in the `sun-web.xml` file. For more information about this element, see "parameter-encoding" on page 197.

- The character encoding specified in the `default-charset` attribute of the `parameter-encoding` element in the `sun-web.xml` file. For more information about this element, see “parameter-encoding” on page 197.

Servlet Response

When processing a servlet response, the server uses the following order of precedence, first to last, to determine the response character encoding:

- The `ServletResponse.setContentType ()` method or the `ServletResponse.setLocale ()` method.
- The default encoding, which is ISO-8859–1.

To specify the character encoding that should be set in the `Content-type` header of the response if the response locale is set using the `ServletResponse.setLocale` method, use the `locale-charset-map` under the `locale-charset-info` element in the `sun-web.xml` file. For more information about this element, see “locale-charset-info” on page 198

Migrating Legacy Servlets

Netscape Enterprise Server/ iPlanet Web Server 4.0 and 4.1 supported the Java Servlet 2.1 specification. This specification did not include web applications. A deployment scheme was developed to make servlet deployment simpler. With the advent of Java web applications (`.war` files) and their deployment descriptors, it is no longer necessary to maintain a proprietary deployment system.

iPlanet Web Server 6.0 supported both types of deployment schemes, but the 4.x implementation (referred to as legacy servlets) was marked as deprecated (See Chapter 8: ?Legacy Servlet and JSP Configuration of the iPlanet Web Server, Enterprise Edition Programmer's Guide to Servlets).

Web Server 6.1 and 7.0 no longer support legacy servlets. The legacy-style properties files for the server you want to migrate (`servlet.properties`, `context.properties`, and `rules.properties`) are removed during migration. For more information about these files, see "Appendix A" in the Sun ONE Web Server 6.1 Administrator's Configuration File Reference.

Because there is no one-to-one mapping for all of the features, legacy servlets cannot be migrated automatically. This section describes the main features involved in migrating legacy servlets to web applications.

This section includes following topics:

- “JSP by Extension” on page 64
- “Servlet by Extension of Servlet by Directory” on page 64

- “Registering Servlets” on page 64

JSP by Extension

In Sun Java System Web Server 7.0, JSP by extension works as it did in previous releases. Any file in the document tree that is named as an extension of `.jsp` is treated as a JSP as long as the Java is turned on for the virtual server.

Servlet by Extension of Servlet by Directory

This feature is not supported in Sun Java System Web Server 7.0. You can deploy a web application to respond to a directory, but all of the servlets must be in the `WEB-INF/classes` directory of the web application. You can no longer copy a servlet in the `.class` file into the document tree and have it run as a servlet or have all of the contents of a directory run as a servlet. The web application will treat only `.class` files as servlets.

Registering Servlets

In the legacy servlet system there was a two-step process of registering servlets (`servlet.properties`) and mapping them to a URL (`rules.properties`). In Sun Java System Web Server 7.0, the servlets must be moved into a web application, and these settings will be maintained in the `web.xml` file of that web application.

EXAMPLE 4–1 Registering Servlets Example

A registered servlet has entries in both the `servlet.properties` and `rules.properties` files.

The following example uses a servlet file called `BuyNow1A.class`, which responds to `/buynow`. It is assumed that the web application is deployed at '/'

The `servlet.properties` file has:

```
servlet.BuyNowServlet.classpath=D:/Netscape/server4/docs/
servlet/buy;D:/Netscape/server4/docs/myclasses
servlet.BuyNowServlet.code=BuyNow1A
servlet.BuyNowServlet.initArgs=arg1=45,arg2=online,arg3="quick shopping"
```

The `rules.properties` file has:

```
/buynow=BuyNowServlet
```

These must be translated to a `web.xml` setting. The `servlet.properties` setting translates into the `servlet` element.

EXAMPLE 4–1 Registering Servlets Example *(Continued)*

The classpath is automated so there is no classpath setting. All classes to be used must be in the `WEB-INF/classes` directory or in a `.jar` file in the `WEB-INF/lib` directory of the web application.

The `servlet-name` element is the part between the dots in the `servlets.properties` file. The code translates to the `servlet-class`, `IntArgs` translate to `init-params`. This entry will translate to:

```
<servlet>
<servlet-name> BuyNowServlet </servlet-name>
<servlet-class> BuyNow1A </servlet-class>
        <init-param>
                <param-value> arg1 </param-name>
                <param-value> 45 </pram-value>
         </init-param>
         <init-param>
                <param-name> arg2 </param-name>
                <param-value> online </param-value>
         </init-param>
         <init-param>
                <param-name> arg3 </param-name>
                <param-value> 'quick shopping" </param-value>
            </init-param>
</servlet>
```

The `rules.properties` entries translate to `servlet-mapping` elements. This entry will translate to:

```
<servlet-mapping>
                    <servlet-name> BuyNowServlet </servlet-name>
                    <url-pattern> /buynow </url-pattern>
    </servlet-mapping>
```

Some other entries in the `servlet.properties` file map to the `web.xml` file. It includes the following.

- `Servlets.startup`: The servlet listed here should have load-on-startup element in it.
- `Servlets.config.reloadInterval`: This translates to the `dynamicreloadinterval` attribute of the `jvm` element in `servlet.xml`. This is an instance-wide setting so it affects all virtual servers and all web applications.
- `Servlet.sessionmgr`: This translates to the `session-manager` element in the `sun-web.xml`.

◆ ◆ ◆ CHAPTER 5

Developing JavaServer Pages

This chapter describes how to use JavaServer Pages (JSP™) as page templates in a Sun Java System Web Server 7.0 application.

This chapter has the following sections:

Introducing JSPs

JSPs are browser pages in HTML or XML. They also contain Java code, which enables them to perform complex processing, conditionalize output, and communicate with other application objects. JSPs in Sun Java System Web Server 7.0 are based on the JSP 2.0 specification.

In a Sun Java System Web Server 7.0 application, you can call a JSP from a servlet to handle the user interaction output, or, since JSPs have the same application environment access as any other application component, you can use a JSP as an interaction destination.

JSPs are made up of JSP elements and template data. Template data is anything not in the JSP specification, including text and HTML tags. For example, the minimal JSP requires no processing by the JSP engine and is a static HTML page.

Sun Java System Web Server 7.0 compiles JSPs into HTTP servlets the first time they are called (or they can be precompiled for better performance). This makes them available to the application environment as standard objects and enables them to be called from a client using a URL.

JSPs run inside the Sun Java System Web Server 7.0 JSP engine, which is responsible for interpreting tags that are specific to JSP and performing the actions they specify in order to generate dynamic content. This content, along with any template data surrounding it, is assembled into an output page and is returned to the caller.

Creating JSPs

You can create JSPs basically the same way you create HTML files. You can use an HTML editor to create pages and edit the page layout. You make a page a JSP by inserting tags that are specific to JSP into the raw source code where needed, and by giving the file a `.jsp` extension. In JSP 2.0 web application resources with an extension of `.jspx` are interpreted as JSP documents.

JSPs specification distinguishes between two types, classic syntax (defined by the JSP specification itself), XML syntax (also referred to as JSP documents). For a summary of the JSP tags you can use, see "JSP Tag Libraries and Standard Portable Tags" on page 72.

JSPs are compiled into servlets, so servlet design considerations also apply to JSPs. JSPs and servlets can perform the same tasks, but each excels at one task at the expense of the other. Servlets are strong in processing and adaptability. However, performing HTML output from them involves many cumbersome `println` statements that must be coded. Conversely, JSPs excel at layout tasks because they are simply HTML files and can be created with HTML editors, though performing complex computational or processing tasks with them is awkward. For information about servlets, see Chapter 4.

Additional JSP design tips are described in the following sections:

- "Designing for Ease of Maintenance" on page 68
- "Designing for Portability" on page 68
- "Handling Exceptions" on page 69

Designing for Ease of Maintenance

Each JSP can call or include any other JSP. For example, you can create a generic corporate banner, a standard navigation bar, and a left-side column table of contents, where each element is in a separate JSP and is included for each page built. The page can be constructed with a JSP functioning as a frameset, dynamically determining the pages to load into each subframe. A JSP can also be included when the it is compiled into a servlet or when a request arrives.

Designing for Portability

JSPs is completely portable between different applications and different servers. A disadvantage is that they have no particular application data knowledge.

Generic JSPs is for portable page elements, such as navigation bars or corporate headers and footers, which are meant to be included in other JSPs. You can create a library of reusable generic page elements to use throughout an application, or even among several applications.

For example, the minimal generic JSP is a static HTML page with no JSP-specific tag. A slightly less minimal JSP contain some Java code that operates on generic data, such as printing the date and time, or that makes a change to the page's structure based on a standard value set in the request object.

Handling Exceptions

If an uncaught exception occurs in a JSP file, Sun Java System Web Server 7.0 generates an exception, usually a 404 or 500 error. To avoid this problem, set the `errorPage` attribute of the page directive. For example,<%@ page other_page_directive_attr_list errorPage=/oops.jsp%> tag.

Compiling JSPs Using the Command-Line Compiler

Sun Java System Web Server 7.0 provides the following ways of compiling JSP 2.0-compliant source files into servlets:

- JSP are automatically compiled at runtime.
- The `jspc` command-line tool, described in this section, allows you to precompile JSPs at the command line.

You must disable dynamic reloading of JSP when deploying a web application archive that has precompiled JSP without the corresponding `jsp` source files. To do this, set the `reload-interval` property to `-1` in the `jsp-config` element of the `sun-web.xml` file. For more information, see "JSP Elements" on page 194.

The `jspc` command-line tool is located under `install_dir/bin` (make sure this directory is in your path). The format of the `jspc` command is as follows:

```
jspc [options] file_specifier
```

The following table shows what `file_specifier` can be in the `jspc` command. The left column lists file specifiers, and the right column lists descriptions of those file specifiers.

TABLE 5–1 File Specifiers for the `jspc` Command

File Specifier	Description
files	One or more JSP files to be compiled.

TABLE 5–1 File Specifiers for the `jspc` Command *(Continued)*

File Specifier	Description
`-webapp` *dir*	A directory containing a web application. All JSPs in the directory and its subdirectories are compiled. You cannot specify a WAR, JAR, or ZIP file; you must first extract it to an open directory structure.

The following table shows the basic options for the `jspc` command. The left column lists the option, and the right column describes what the option does.

TABLE 5–2 Basic `jspc` Options

Option	Description
`-help`	Enables quiet mode (same as `-v0`). Only fatal error messages are displayed.
`-v`	Verbose Mode.
`-d` *dir*	Specifies the output directory for the compiled JSPs. Package directories are automatically generated based on the directories containing the uncompiled JSPs. The default top-level directory is the directory from which `jspc` is invoked.
`-l`	Specifies the name of the JSP on failure.
`-s`	Specifies the name of the JSP on success.
`-p` *name*	Specifies the name of the target package for all specified JSPs, overriding the default package generation performed by the `-d` option.
`-c` *name*	Specifies the target class name of the first JSP compiled. Subsequent JSPs are unaffected.
`-mapped`	Generates separate `write()` calls for each HTML line in the JSP.
`die(#)`	Generates an error return code (#) on fatal errors (default 1).
`-uribase` *dir*	Specifies the URI directory to which compilations are relative. Applies only to explicitly declared JSP files. This is the location of each JSP file relative to the `uriroot`. If this cannot be determined, the default is `/`.
`-uriroot` *dir*	A directory containing a web application. All JSPs in the directory and its subdirectories are compiled. You cannot specify a WAR, JAR, or ZIP file; you must first extract it to an open directory structure.
`-compile`	Compiles the generated servlets.
`-genclass`	Generates class files in addition to Java files.
`-webinc` *file*	Creates a partial servlet mappings in the file.

TABLE 5–2 Basic `jspc` Options (Continued)

Option	Description
`-web.xml` *file*	Creates a complete `web.xml` in the file.
`-ieplugin` *clsid*	Java plug-in `classid` for Internet Explorer.
`classpath` *path*	Override the `java.class.path` system property
`xpoweredBy`	Add X-Powered-By response header.
`-trimSpaces`	Trim spaces in text templates between actions and directives.
`-smap`	Generates SMAP info for JSR45 debugging.
`-dumpsmap`	Dump SMAP info for JSR45 debugging into a file.
`-compilerSourceVM` *release*	Provides source compatibility with specified JDK release.
`-compilerTargetVM`*release*	Generates class files for specified VM version.

For example, this command compiles the `hello` JSP file and writes the compiled JSP under `hellodir`:

```
jspc -d hellodir -genclass hello.jsp
```

This command compiles all of the JSP files in the web application under `webappdir` into class files under `jspclassdir`:

```
jspc -d jspclassdir -genclass -webapp webappdir
```

To use either of these precompiled JSPs in a web application, put the classes under `hellodir` or `jspclassdir` into a JAR file, place the JAR file under `WEB-INF/lib`, and set the `reload-interval` property to `-1` in the `sun-web.xml` file.

Package Names Generated by the JSP Compiler

When a JSP is compiled, a package is created for it. The package name starts with `jspc`. For example, the generated package name for `~/SOURCE/JSP/myjsps/hello.jsp` will be precompiled as `jspc -webapp ~/SOURCE -d ~/test1/test2/test3`. The generated servlet will be located in `~/test1/test2/test3/org/apache/jsp/JSP/myjsps/hello_jsp.java` and defined in `org.apache.jsp.JSP.myjsps`. Notice how `hello.jsp` is derived from the directory in which the JSP is located.

In another example, the same hello `jsp` is pre compiled using the –p option, and is precompiled as `jspc -webapp ~/SOURCE -d ~/test1/test2/test3 -p app1.app2.app3`. The generated servlet will be located in `~/test1/test2/test3/app1/app2/app3/JSP/myjsps/hello_jsp.java` and defined inside package `app1.app2.app3.JSP.myjsps`. Notice how the package specified via the `-p` option (`app1.app2.app3`) overrides the standard `org.apache.jsp`, but does not affect the package derived from the directory in which the JSP is located. Also notice how the `-d` option does not have any impact on the generated package name.

Other JSP Configuration Parameters

For information about the various JSP configuration parameters you can use, see the section "jsp-config" on page 194. The JSP compiler uses the default values for parameters that are not included in the file.

Debugging JSPs

When you use Sun Java Studio Enterprise 8.1 to debug JSPs, you can set breakpoints in a JSP.

For information about setting up debugging in Sun Java Studio Enterprise 8.1, see "Using Developer tools for Debugging" on page 153.

JSP Tag Libraries and Standard Portable Tags

Sun Java System Web Server 7.0 supports tag libraries and standard portable tags. For more information about tag libraries, see the JSP 2.0 specification at:

`http://java.sun.com/products/jsp/download.html`

JSP Cache Tags

JSP cache tags allow you to cache JSP page fragments within the Java engine. Each can be cached using different cache criteria. For example, suppose you have page fragments to view stock quotes, and weather information. The stock quote fragment can be cached for 10 minutes, the weather report fragment for 30 minutes, and so on.

For more information about response caching as it pertains to servlets, see "Caching Servlet Results" on page 56.

JSP caching uses the custom tag library support provided by JSP 2.0. JSP caching is implemented by a tag library packaged into the *install_dir*`/lib/.jar` file, which is always on the server classpath. The `sun-web-cache.tld` tag description file is in *install_dir*`/lib/tlds` directory and can be copied into the `WEB-INF` directory of your web application.

JSP caching is available in three different scopes: request, session, and application. The default is application.

To use a cache with the request scope, a web application must specify the `com.sun.appserv.web.taglibs.cache.CacheRequestListener` in its deployment descriptor, as follows:

```
<listener>
   <listener-class>
      com.sun.appserv.web.taglibs.cache.CacheRequestListener
   </listener-class>
</listener>
```

To cache with the session scope, to use a cache in session scope, a web application must specify the `com.sun.appserv.web.taglibs.cache.CacheSessionListener` in its deployment descriptor, as follows:

```
<listener>
   <listener-class>
      com.sun.appserv.web.taglibs.cache.CacheSessionListener
   </listener-class>
</listener>
```

To use a cache with the application scope, a web application does not need to specify any listener, the `com.sun.appserv.web.taglibs.cache.CacheContextListener` is already specified in the `sun-web-cache.tld` file.

The JSP cache tags are as follows:

- "`cache`" on page 73
- "`flush`" on page 75

cache

The `cache` tag allows you to cache fragments of your JSP pages. It caches the body between the beginning and ending tags according to the attributes specified. The first time the tag is encountered, the body content is executed and cached. Each subsequent time it is run, the cached content is checked to see if it needs to be refreshed and if so, it is executed again, and the cached data is refreshed. Otherwise, the cached data is served.

Attributes

The following table describes attributes for the `cache` tag. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE 5–3 cache Attributes

Attribute	Default	Description
key	*servletPath_suffix*	(Optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry. If no key is specified, a number is generated according to the position of the tag in the page.
timeout	60s	(Optional) The time in seconds after which the body of the tag is executed and the cache is refreshed. By default, this value is interpreted in seconds. To specify a different unit of time, add a suffix to the timeout value as follows: s for seconds, m for minutes, h for hours,and d for days. For example, 2h specifies two hours.
nocache	false	(Optional) If set to true, the body content is executed and served as if there were no cache tag. This offers a way to programmatically decide whether the cached response should be sent or whether the body must be executed, though the response is not cached.
refresh	false	(Optional) If set to true, the body content is executed and the response is cached again. This lets you programmatically refresh the cache immediately, regardless of the timeout setting.
scope	application	Specifies the scope of the cache. Valid values are request, session, and application.

Example

The following example represents a cached JSP page:

```
<%@ taglib prefix="mypfx" uri="/com/sun/web/taglibs/cache" %>
            <%
           String cacheKey = null;
            if (session != null)
          cacheKey = (String)session.getAttribute("loginId");
           // check for nocache
               boolean noCache = false;
           String nc = request.getParameter("nocache");
           if (nc != null)
          noCache = "true";
               // force reload
               boolean reload=false;
               String refresh = request.getParameter("refresh");
               if (refresh != null)
             reload = true;
             %>
             <mypfx:cache key="<%= cacheKey %/>" nocache="<%= noCache %/>"
       refresh="<%= reload %/>" timeout="10m"/>
```

```
             <%
                  String page = request.getParameter("page");
            if (page.equals("frontPage") {
            // get headlines from database
             } else {
       .....
              %>
                 </mypfx:cache/>
                             <mypfx:cache timeout="1h"/>
                 <h2> Local News </h2>
             <%
                // get the headline news and cache them
             %>
                 </mypfx:cache/>
```

flush

Forces the cache to be flushed. If a key is specified, only the entry with that key is flushed. If no key is specified, the entire cache is flushed.

Attributes

The following table describes attributes for the `flush` tag. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE 5–4 `flush` Attributes

Attribute	Default	Description
`key`	*servletPath_suffix*	(Optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry. If no key is specified, a number is generated according to the position of the tag in the page.
`scope`	`application`	Specifies the scope of the cache. Valid values are `request`, `session`, and `application`.

Examples

To flush the entry with `key="foobar"`:

```
<mypfx:flush key="foobar"/>
```

To flush the entire cache:

```
<% if (session != null && session.getAttribute("clearCache") != null) { %/>
     <mypfx:flush />
 <% } %/>
```

JSP Search Tags

Sun Java System Web Server 7.0 includes a set of JSP tags that can be used to customize the search query and search results pages in the search interface. This section describes the tags and how they are used.

The search tag library is packaged into the *install_dir*`/lib/webserv-rt.jar` file, which is always on the server classpath. The `sun-web-search.tld` tag description file can be found in the *install_dir*`/lib/tlds` directory, and can be copied into the `WEB-INF` directory of your web application.

The search tags are as follows:

- "`searchForm`" on page 77
- "CollElem" on page 78
- "collection" on page 79
- "collItem" on page 79
- "queryBox" on page 80
- "submitButton" on page 80
- "formAction" on page 81
- "formSubmission" on page 82
- "formActionMsg" on page 82
- "Search" on page 83
- "resultIteration" on page 83
- "Item" on page 84
- "resultStat" on page 84
- "resultNav" on page 84

Note – The Sun Java System Web Server 7.0 search feature is I18n compliant, and supports multiple character encoding schemes in the same collection. Custom JSPs that expose search can be in any encoding.

---

searchForm

Function

Constructs an HTML form that contains default, hidden form elements such as the current search result index and number of records per page by default. The default names for the form, index, and number of records are `searchform`, `si`, and `ns`.

Attributes

The `searchForm` tag uses the following attributes:

- Name. Specifies the name of the form. The default is `searchform`. The name of a form is the identifier for all other tags.
- Action. (Optional) Specifies the form action.
- Method. (Optional) Specifies the method of submission, GET or POST. The default is GET.
- elemStart. (Optional) Specifies the name of the hidden `Start` element. If not specified, the default `si` will be used.
- Start. (Optional) An integer indicating the starting index for displaying the search result. The default is 1.
- elemNumShown. (Optional) The name of the `nShown` element. If not specified, the default `ns` is used.
- numShown. (Optional) An integer indicating the number of results to be shown per page. The value of the attribute will be retrieved by requesting parameter `elemNumShown`. The default is 10.

Usage

```
<s1ws:form action="results.jsp" />
...
</s1ws:searchForm>
```

This example creates an HTML form tag `<form name="searchform" action="results.jsp" method="GET">`, along with two hidden input boxes:

- A hidden input box for starting the index named `si` with a value from the `si` parameter or default 1
- A hidden input box for the number of records per page named `ns` with a value from the `ns` parameter or default 20.

CollElem

Function

Creates a hidden inputbox or select box, or multiple checkboxes, depending on the attribute input. If there is only one collection, the tag will create a hidden inputbox by default.

Attributes

The `CollElem` tag uses the following attributes:

- Name. Specifies the name of the form element created. The default is `c`.
- Items. (Optional) A comma-delimited string representing search collections available. The tag retrieves all collections available on the server if the attribute is empty.
- Type. (Optional) The type of form element used for displaying collections. Valid options are `hidden`, `select`, and `checkbox`. The default value is `hidden` if there is only one collection, and `checkbox` if there are multiple collections.
- Rows. (Optional) Represents size if the type is `select`, or the number of rows. The default behavior is to satisfy the `Cols` attribute first. That is, the collections will be listed in columns as specified by the `Cols` attribute.
- Cols. Represents number of columns and is only required if type is `checkbox`. If `Cols` and `Rows` are not specified, the collections will be listed horizontally.
- Defaults. Specifies a comma-delimited string containing 1s or 0s indicating the selection status of the search collections. An item is selected if 1, and not selected if 0. If there is a form action, these values will be retrieved from the form elements.
- cssClass. (Optional) The class name that will be used in every HTML tag created in this tag. This attribute is particularly useful when the type is `checkbox`, since an HTML table will be used for the layout. See the sample code below for details.

Usage

```
<s1ws:collElem type="checkbox" cols="2" values="1,0,1,0" cssClass="body" />
```

This example creates checkboxes in 2 columns with a default name "c," with the first and third items selected. Fonts and any other HTML styles are defined in css classes "body," which includes `tr.body`, `td.body`, and `input.body`.

collection

Function

Retrieves the name of the search collections on the server, iterates through, and passes each of them to the `collectionitem` tag. Use this tag along with collection item tags so they can write their own HTMLs.

Attributes

items. (Optional) A comma-delimited string representing the search collections available. The tag retrieves all collections available on the server if the attribute is empty.

Usage

```
<table border=0>
                <s1ws:collection>
                <tr><td><input type=checkbox name="c"
           value="<s1ws:collItem property="name" />">
        <s1ws:collItem property="display name" /
                ></td></tr>
            </s1ws:collection>
</table>
```

The above code will display all collections with checkboxes.

```
<select name=elementname>
<s1ws:collection>
<option value="<s1ws:collItem property="name" />">
<s1ws:colleItem property="display name" />
</s1ws:collection>
</select>
```

This iterates through the available collections and passes each item to the collection item tag, which in turn displays the name and display name of the item.

collItem

Function

Displays the name and label of one collection item. Must be used inside the `collection` tag.

Attributes

Property. Specifies a keyword indicating which property the tag should output. Valid inputs include `name`, `display name`, and `description`. Default is `name`.

Usage

```
<select name=elementname>
            <s1ws:collection>
            <option value="<s1ws:collItem property="name" />">
     <s1ws:collItem property="display name" />
            </s1ws:collection>
            </select>
```

This iterates through the available collections and passes each item to the collection item tag, which in turn displays the name and display name of the item.

queryBox

Function

Creates an input box.

Attributes

The `queryBox` tag uses the following attributes:

- name. (Optional) The name of the inputbox. The default is "qt."
- default. (Optional) The default value for the query box. If the form is submitted, its value will be set using what has been submitted.
- size. (Optional) The size of the inputbox. The default is 50.
- maxlength. (Optional) The `maxlength` of the inputbox. The default is 50.
- cssClass. (Optional) The CSS class.

Usage

```
<s1ws:queryBox size="30"/>
```

This example creates an inputbox with default name "qt" and `size=30`.

submitButton

Function

Creates a submit button.

Attributes

The `submitButton` tag uses the following attributes:

- name. (Optional) The name of the button. The default is "sb."
- default. (Optional) The default value of the button, which will be "search."
- `cssClass`. (Optional) The CSS class name.
- style. The CSS style.
- image. The optional image for the button.

Usage

```
<s1ws:submitButton cssClass="navBar" style="padding: 0px; margin: 0px; width: 50px">
```

This example creates a submit button with default name "sb."

formAction

Function

Handles form action. It retrieves all elements on the search form, validates that there is at least one collection selected and the query is not empty, and passes the values on to search and results tags as parents or through the page context.

Attributes

The `formAction` tag uses the following attributes:

- formId. Specifies the name of the form. If not specified, the default form "searchForm" will be used.
- ElemColl. (Optional) The name of the `Collection` element. The default name "c" is used.
- elemQuery. (Optional) The name of the `Start` element. The default name "qt" is used.
- elemStart. (Optional) The name of the `Start` element. The default name "si" is used.
- elemNumShown. (Optional) The name of the `numShown` element. The default name "ns" is used.

Usage

```
<s1ws:formAction />
```

formSubmission

Function

Tests if the form submission is successful.

Attributes

The `formsubmission` tag uses the following attributes:

- formId. Specifies the name of the form in question. It must be the same as that for `<formAction>`.
- success. Indicates if the form submission is successful. The values `true` or `yes` represents successful action. All other inputs will be rendered as failure.

Usage

```
<s1ws:formSubmission success="true" >
         <s1ws:search>
         ...
   </s1ws:formSubmission>
```

formActionMsg

Function

Prints out an error message from `formAction`, if any.

Attributes

The `formActionMsg` tag uses the following attributes:

- formId. (Optional) Specifies the name of the form in question. If not specified, the default ID is `searchForm`.
- elem. (optional) Specifies the name of the element. Valid inputs are `query` and `collection`. When specified, the tag returns an error message, if any, regarding the element. Otherwise, it prints out all of the error messages generated.

Usage

```
<FormActionMsg elem="query">
```

This tag will display a message `query text not specified` if a query is not submitted.

The message is printed from the form action where the tag is placed.

Search

Function

Executes a search query and returns search results. The search tag retrieves a query string and collection(s) from either a `form` parent tag or the `query` and `collection` attributes. The search results are saved in the page context with a page or session scope.

Attributes

The `search` tag uses the following attributes:

- formId. Specifies the name of the form used for the search. The default form will be used if the attribute is left empty. If this tag is used without a form, this attribute must be set to provide an identifier for the `resultIterate` tag.
- collection. (Optional) A comma-delimited string representing collection(s) used for a search. If there is a form action, this attribute is ignored and the values are retrieved by requesting the `collection` element.
- query. (Optional) Specifies the query text. If not provided, it is retrieved from the `query` element.
- scope. Specifies an integer indicating the scope of the search results. The value 1, which is the default, indicates page scope, and 2 indicates session scope.

Usage

```
<s1ws:search />
```

This search tag uses the default parameters and executes a search. The search results will be saved in `pageContext` with a page scope.

```
<s1ws:search Collection="col1, col2" Query="Java Web Service" scope="2" />
```

This search tag will execute a search in `col1` and `col2` using `"Java Web Service"` as the query string. The search results will be saved in `pageContext` with a session scope.

resultIteration

Function

Retrieves the search results from either the parent search tag or the page context. The tag is responsible for iterating through the results and passing the `searchitems` to the item tags.

Attributes

The `resultIteration` tag uses the following attributes:

- formId. Specifies the name of the form associated with the search results. The default form will be used if the attribute is left empty. If this tag is used without a form, this attribute must be set as a reference to the search tag.
- start. Specifies an integer representing the starting position in the search results. The default is 0. The value is retrieved from the parent `<formAction>` tag or `pageContext`, or the parameter value.
- numShown. Specifies an integer indicating the number of results to be shown in one page. The default is 20. The value is retrieved from the parent `<formAction>` tag or `pageContext`.

Item

Function

Retrieves a `searchitem` from the `Results` parent tag and outputs its properties as specified by the `property` attribute.

Attributes

Property. Specifies a case-sensitive string representing field names in a search item, such as title, number, score, filename, URL, size, and date.

resultStat

Function

Returns numbers indicating number of records returned and the range currently displayed.

Attributes

The `resultStat` tag uses the following attributes:

- formId. Specifies the name of the form associated. The default form will be used if the attribute is left empty. If this tag is used without a form, this attribute must be set as a reference to the search tag.
- type. Specifies the type of statistics data. Valid inputs are "start," "end," "range" (for example, 1-20), and "total."

resultNav

Function

Creates a navigation bar.

Attributes

The `resultNav` tag uses the following attributes:

- `formId`. Specifies the name of the form associated with the navigation bar. The default form will be used if the attribute is left empty. If this tag is used without a form, this attribute must be set as a reference to the search tag.
- type. Specifies the type of navigation bar. Valid values are `full`, `previous`, and `next`. A full navigation bar looks like this: `previous 1 2 3 4 5 6 7 8 9 10 next`, where `5` is currently selected. The number of `page number` links is determined by the `offset` attribute and the number of pages available.
- caption. Only necessary if the type is `previous` or `next` and the default text is not needed. `caption` can be any HTML.
- offset. Specifies the number of page links around the selected page. For example, if `offset=2`, the above sample bar above will look like this: `previous 3 4 5 6 7 next`. Only required for type "full."

JSP Internationalization Issues

This section covers internationalization as it applies to the JSPs.

JSPs

A JSP page uses a character encoding. For valid encodings to use, see: `http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html`

The encoding can be described explicitly using the `pageEncoding` attribute of the `page` directive. The character encoding defaults to the encoding indicated in the `contentType` attribute of the `page` directive, if it is given.

◆ ◆ ◆ CHAPTER 6

Session Managers

Session objects maintain state and user identity across multiple page requests over the normally stateless HTTP protocol. A session persists for a specified period of time, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session either by using cookies or by rewriting URLs. Servlets can access the session objects to retrieve state information about the session.

This chapter describes sessions and session managers, and has the following sections:

- "Introducing Sessions" on page 87
- "How to Use Sessions" on page 89
- "Session Managers" on page 93

Introducing Sessions

The term *user session* refers to a series of user application interactions that are tracked by the server. Sessions are used for maintaining user specific state, including persistent objects (such as handles to database result sets) and authenticated user identities, among many interactions. For example, a session can be used to track a validated user login, followed by a series of directed activities for a particular user.

The session itself resides in the server. For each request, the client transmits the session ID in a cookie or, if the browser does not allow cookies, the server automatically writes the session ID into the URL.

The Sun Java System Web Server 7.0 supports the servlet standard session interface, called `HttpSession`, for all session activities.

This section includes the following topics:

- "Sessions and Cookies" on page 88
- "Sessions and URL Rewriting" on page 88
- "Sessions and Security" on page 88

Note – As of Sun Java System Web Server 7.0, `form-login` sessions are no longer supported. You can use single sign-on sessions instead.

Sessions and Cookies

A cookie is a small collection of information that can be transmitted to a calling browser, which retrieves it on each subsequent call from the browser so that the server can recognize calls from the same client. A cookie is returned with each call to the site that created it, unless it expires.

Sessions are maintained automatically by a session cookie that is sent to the client when the session is first created. The session cookie contains the session ID, which identifies the client to the server on each successive interaction. If a client does not support or allow cookies, the server rewrites the URLs where the session ID appears in the URLs from that client.

You can configure whether and how sessions use cookies. For more information, see the "session-properties" on page 174 and "cookie-properties" on page 175 elements in the `sun-web.xml` file, described in Chapter 9.

Sessions and URL Rewriting

You can configure whether sessions use URL rewriting. For more information, see the `sun-web.xml` element "session-properties" on page 174.

Sessions and Security

The Sun Java System Web Server 7.0 security model is based on an authenticated user session. Once a session has been created, the application user is authenticated (if authentication is used) and logged into the session.

Additionally, you can specify that a session cookie is only passed on a secured connection (that is, HTTPS), so the session can only remain active on a secure channel.

For more information about security, see Chapter 8.

How to Use Sessions

To use a session, first create a session using the `HttpServletRequest` method `getSession()`. Once the session is established, examine and set its properties using the provided methods. If desired, set the session to time out after being inactive for a defined time period, or invalidate it manually. You can also bind objects to the session, which store them for use by other components.

This section includes the following topics:

Creating or Accessing a Session

To create a new session or gain access to an existing session, use the `HttpServletRequest` method `getSession()`, as shown in the following example:

```
HttpSession mySession = request.getSession();
```

`getSession()` returns the valid session object associated with the request, identified in the session cookie that is encapsulated in the request object. Calling the method with no arguments creates a session if one does not already exist that is associated with the request. Additionally, calling the method with a Boolean argument creates a session only if the argument is `true`.

The following example shows the `doPost()` method from a servlet that only performs the servlet's main functions, if the session is present. Note that the false parameter to `getSession()` prevents the servlet from creating a new session if one does not already exist:

```
public void doPost (HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException
 {
        if ( HttpSession session = req.getSession(false) ) {
           // session retrieved, continue with servlet operations
        }
        else{
           // no session, return an error page
        }
}
```

Note – The `getSession()` method should be called before anything is written to the response stream.

For more information about `getSession()`, see the Java Servlet 2.4 specification.

Examining Session Properties

Once a session ID has been established, use the methods in the `HttpSession` interface to examine session properties, and the methods in the `HttpServletRequest` interface to examine request properties that relate to the session.

The following table shows the methods used to examine session properties. The left column lists `HttpSession` methods, and the right column lists descriptions of these methods.

TABLE 6–1 `HttpSession` Methods

HttpSession Method	Description
`getCreationTime()`	Returns the session time in milliseconds since January 1, 1970, 00:00:00 GMT.
`getId()`	Returns the assigned session identifier. An HTTP session's identifier is a unique string that is created and maintained by the server.
`getLastAccessedTime()`	Returns the last time the client sent a request carrying the assigned session identifier (or `-1` if it's a new session) in milliseconds since January 1, 1970, 00:00:00 GMT.
`isNew()`	Returns a Boolean value indicating that the session is new. It is a new session if the server has created it and the client has not sent a request to it. This means the client has not acknowledged or joined the session and may not return the correct session identification information when making its next request.

For example:

```
String mySessionID = mySession.getId();
     if ( mySession.isNew() ) {
         log.println(currentDate);
        log.println("client has not yet joined session " + mySessionID);
     }
```

The following table shows the methods used to examine servlet request properties. The left column lists `HttpServletRequest` methods, and the right column lists descriptions of these methods.

TABLE 6–2 `HttpServletRequest`Methods

HttpServletRequest Method	Description
`getRequestedSessionId()`	Returns the session ID specified with the request. This value may differ from the session ID in the current session if the session ID given by the client is invalid and a new session was created. Returns null if the request does not have a session associated with it.
`isRequestedSessionIdValid()`	Checks if the request is associated to a currently valid session. If the session requested is not valid, it is not returned through the `getSession()` method.
`isRequestedSessionIdFromCookie()`	Returns `true` if the request's session ID provided by the client is a cookie, or `false` otherwise.
`isRequestedSessionIdFromURL()`	Returns `true` if the request's session ID provided by the client is a part of a URL, or `false` otherwise.

For example:

```
if ( request.isRequestedSessionIdValid() ) {
        if ( request.isRequestedSessionIdFromCookie() ) {
           // this session is maintained in a session cookie
         }
        // any other tasks that require a valid session
      } else {
        // log an application error
      }
```

Binding Data to a Session

You can bind objects to sessions to make them available across multiple user interactions.

The following table shows the `HttpSession` methods that provide support for binding objects to the session object. The left column lists `HttpSession` methods, and the right column lists descriptions of these methods.

TABLE 6–3 `HttpSession` Methods

HttpSession Method	Description
`getAttribute()`	Returns the object bound to a given name in the session, or null if there is no such binding.
`getAttributeNames()`	Returns an array of names of all attributes bound to the session.
`setAttribute()`	Binds the specified object into the session with the given name. Any existing binding with the same name is overwritten. For an object bound into the session to be distributed, it must implement the serializable interface.
`removeAttribute()`	Unbinds an object in the session with the given name. If there is no object bound to the given name, this method does nothing.

Binding Notification with `HttpSessionBindingListener`

Some objects require you to know when they are placed in or removed from a session. To obtain this information, implement the `HttpSessionBindingListener` interface in those objects. When your application stores or removes data with the session, the servlet engine checks whether the object being bound or unbound implements `HttpSessionBindingListener`. If it does, the Sun Java System Web Server 7.0 notifies the object under consideration, through the `HttpSessionBindingListener` interface, that it is being bound into or unbound from the session.

Invalidating a Session

Direct the session to invalidate itself automatically after being inactive for a defined time period. Alternatively, invalidate the session manually with the `HttpSession` method `invalidate()`.

Invalidating a Session Manually

To invalidate a session manually, simply call the following method:

```
session.invalidate();
```

All objects bound to the session are removed.

Setting a Session Timeout

Session timeout is set using the `session-timeout` element in the `web.xml` deployment descriptor file. For more information, see the Java Servlet 2.4 specification.

Session Managers

Sun Java System Web Server 7.0 provides the following session management options, which are described in this section:

- "memory" on page 93, the default session manager
- "file" on page 94, a provided manager that store sessions on the file system
- "IWS60" on page 95, a provided session manager that allows backward compatibility with any custom session managers you may have created using Sun Java System Web Server 7.0
- "MMap(UNIX Only)" on page 101, a provided persistent memory map (mmap) file-based session manager that works in both single-process and multi-process mode

memory

The memory is the default memory based session manager.

Enabling memory

You can specify memory explicitly to change its default parameters. To do so, edit the sun-web.xml file for the web application as in the following example. Note that persistence-type must be set to memory.

```
<sun-web-app>
         ...
        <session-config>
           <session-manager persistence-type="memory">
              <manager-properties>
                 <property name="reapIntervalSeconds" value="20" />
              </manager-properties>
           </session-manager>
            ...
        </session-config>
        ...
</sun-web-app>
```

For more information about the sun-web.xml file, see Chapter 9.

Manager Properties for Memory

The following table describes manager-properties properties for the memory based session manager. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE 6–4 manager-properties for `memory`

Property Name	Default Value	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions. Setting this value lower than the frequency at which session data changes is recommended. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed web site, or you could lose the last few hits each time you restart the server.
`maxSessions`	-1	Specifies the maximum number of active sessions, or `-1` (the default) for no limit.
`sessionFilename`	`SESSIONS`	Specifies the absolute or relative path name of the file in which the session state is preserved between application restarts, if preserving the state is possible. A relative path name is relative to the temporary directory for this web application.

file

The `file` is another file-system-based session manager provided with Sun Java System Web Server 7.0. For session persistence, `file` can use a file to which each session is serialized. You can also create your own persistence mechanism.

Enabling file

You can specify `file` explicitly to change its default parameters. To do so, edit the `sun-web.xml` file for the web application as in the following example. Note that `persistence-type` must be set to `file`.

```
<sun-web-app>
        ...
        <session-config>
           <session-manager persistence-type=”file”>
              <manager-properties>
                 <property name=reapIntervalSeconds value=20 />
              </manager-properties>
              <store-properties>
                 <property name=directory value=sessions />
              </store-properties>
           </session-manager>
           ...
        </session-config>
        ...
</sun-web-app>
```

For more information about the `sun-web.xml` file, see Chapter 9.

Manager Properties for file

The following table describes `manager-properties` properties for the file session manager. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE 6–5 manager-properties for file

Property Name	Default Value	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions. Setting this value lower than the frequency at which session data changes is recommended. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed web site, or you could lose the last few hits each time you restart the server.
`maxSessions`	-1	Specifies the maximum number of active sessions, or `-1` (the default) for no limit.

IWS60

The `IWS60` session manager ensures backward compatibility with any 6.0 session managers that you may have created.

`IWS60` works in both single-process and multi-process mode. It can be used for sharing session information across multiple processes possibly running on different machines.

Note – The `MaxProcs` directive in the `magnus.conf` file determines whether the server is running in single-process or multi-process mode. If the value of `MaxProcs` is higher than `1` and no session manager is configured, then by default the session manager used is the `IWS60` with file-based persistence. The `Maxprocs` is deprecated in Web Server 7.0.

For more information on the `MaxProcs` directive, see the Sun Java System Web Server 7.0 *Developer's Guide.*

For session persistence, `IWS60` can use a database or a distributed file system (DFS) path that is accessible from all servers in a server farm. Each session is serialized to the database or distributed file system. You can also create your own persistence mechanism.

If Sun Java System Web Server 7.0 is running in single-process mode, then by default, no session persistence mode is defined and therefore sessions are not persistent.

If Sun Java System Web Server 7.0 is running in multi-process mode, sessions are persistent by default. If a persistence mode is not defined, `IWS60` uses a DFS.

Multi-process mode is supported only on UNIX platforms. All multi-process mode features of `IWS60` are ignored on Windows.

Enabling `IWS60`

You can enable `IWS60` to change its default parameters. You can also enable `IWS60` for a particular context if the server is running in single-process mode. To do so, edit the `sun-web.xml` file for the web application as in the following example. The `persistence-type` must be set to `s1ws60`.

```
<sun-web-app>
   ...
   <session-config>
    <session-manager persistence-type="s1ws60">
     <manager-properties>
      <property name="classname" value="com.iplanet.server.http.
                                       session.IWSSessionManager"/>
      // other manager-related properties
     </manager-properties>
    </session-manager>
     ...
   </session-config>
   ...
</sun-web-app>
```

In the case of persistent sessions:

```
<sun-web-app>
        ...
        <session-config/>
         <session-manager persistence-type="s1ws60">
          <manager-properties>
           <property name="classname" value="com.iplanet.server.http.
                                       session.IWSSessionManager"/>
       // other manager-related properties
    </manager-properties>
    <store-properties>
     <property name="classname" value="com.iplanet.server.http.
                                               session.FileStore"/>
     <property name="directory" value="<directory name to store the_
persistant_sessions>"/>
       // other store-related properties
     </store-properties>
    </session-manager>
         ...
   </session-config>
       ...
        </sun-web-app>
```

For more information about the `sun-web.xml` file, see Chapter 9.

Manager Properties for IWS60

The following table describes `manager-properties` properties for the `IWS60` session manager. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE 6–6 manager-properties for `IWS60`

Property Name	Default Value	Description
`maxSessions`	`1000`	The maximum number of sessions maintained by the session manager at any given time. The session manager refuses to create any more new sessions if there are already `maxSessions` number of sessions present at that time.
`timeOut`	`1800`	The amount of time in seconds after a session is accessed by the client before the session manager destroys it. Those sessions that is not accessed for at least `timeOut` seconds are destroyed by the reaper method. The `session-timeout` parameter specified in `web.xml` will not be effective when `IWS60` is used.
`reapInterval`	`600`	The amount of time in seconds that the `SessionReaper` thread sleeps before calling the reaper method again.
`maxLocks`	`10`	The number of cross-process locks to use for synchronizing access to individual sessions across processes. The default value is used if the value `0` is specified. This parameter is ignored in single-process mode.

TABLE 6–6 manager-properties for IWS60 *(Continued)*

Property Name	Default Value	Description
session-data-store		The name of the class that determines the means of session persistence. The classes supplied with Sun Java System Web Server 7.0 are: ■ com.iplanet.server.http.session.JdbcStore ■ com.iplanet.server.http.session.FileStore If you do not specify the session-data-store parameter, sessions are not persistent in single-process mode, and FileStore is the default in multi-process mode. The JdbcStore and FileStore classes are subclasses of the session-data-store class. You can create your own class that implements session persistence by extending SessionDataStore.
session-failover-enabled		Specifies whether sessions are reloaded from the persistent store for every request, and always forced to true in multi-process mode. Applicable only if the session-data-store parameter is set to the JdbcStore or FileStore class.
session-data-dir	(this should all be on one line) *server_root*/server_id/SessionData /virtual_server_id/web_app_URI	The directory in which session data for all servers and web applications is kept. Applicable only if the session-data-store parameter is set to the FileStore class.
provider	sun.jdbc.odbc.JdbcOdbcDriver	The JDBC driver. For more information about the JDBC API, see the following web site: http://java.sun.com/products/jdbc/index.jsp Applicable only if the session-data-store parameter is set to the JdbcStore class.
url	jdbc:odbc:LocalServer	Specifies the data source. Applicable only if the session-data-store parameter is set to the JdbcStore class.

TABLE 6–6 manager-properties for IWS60 *(Continued)*

Property Name	Default Value	Description
table	sessions	Name of the SQL table that store sessions. Applicable only if the session-data-store parameter is set to the JdbcStore class.
username	none	The login user name for the database. Applicable only if the session-data-store parameter is set to the JdbcStore class.
password	none	The login password for the database. Applicable only if the session-data-store parameter is set to the JdbcStore class.
reaperActive	true	When set to true, the session manager runs session reaper to remove expired sessions from the database. The default is true. Only one server in the cluster should run the reaper. Applicable only if the session-data-store parameter is set to the JdbcStore class.
accessTimeColumn	AccessTime	The name of the column that holds the last access time in minutes. The SQL type is NUMERIC(9). Applicable only if the session-data-store parameter is set to the JdbcStore class.
timeOutColumn	TimeOut	The name of the column that holds the session timeout in minutes. The SQL type is NUMERIC(9). Applicable only if the session-data-store parameter is set to the JdbcStore class.
sessionIdColumn	SessionID	The name of the column that holds the session ID. The SQL type is VARCHAR(100). Applicable only if the session-data-store parameter is set to the JdbcStore class.
valueColumn	Value	The name of the column that holds the session object. The SQL type is VARBINARY(4096). This column must be large enough to accommodate all of your session data. Applicable only if the session-data-store parameter is set to the JdbcStore class.

TABLE 6–6 manager-properties for IWS60 *(Continued)*

Property Name	Default Value	Description
lookupPool	4	The number of dedicated connections that perform lookup operations on the database. For higher performance, use a precompiled SQL statement for each of these connections. Applicable only if the session-data-store parameter is set to the JdbcStore class.
insertPool	4	The number of dedicated connections that perform insert operations on the database. Each of these connections would have a precompiled SQL statement for higher performance. Applicable only if the session-data-store parameter is set to the JdbcStore class.
updatePool	4	The number of dedicated connections that perform update operations on the database. For higher performance, use a precompiled SQL statement for each of these connections. Applicable only if the session-data-store parameter is set to the JdbcStore class.
deletePool	2	The number of dedicated connections that perform delete operations on the database. For higher performance, use a precompiled SQL statement for each of these connections. Applicable only if the session-data-store parameter is set to the JdbcStore class.

Note – Prior to using JdbcStore, you must create the table in which the session information is stored. The name of the table is specified by the table parameter, and the table's four columns are specified by the accessTimeColumn, timeOutColumn, sessionIdColumn, and valueColumn parameters.

Note – FileStore, JdbcStore, IWSSessionManager, IWSHttpSession, IWSHttpSessionManager, and SessionDataStore have been deprecated in Sun Java System Web Server 7.0.

Source Code for IWS60

The `IWS60` session manager creates an `IWSHttpSession` object for each session. The source files for `IWSSessionManager.java` and `IWSHttpSession.java` are in the *install_dir*/`lib` directory. The source code files for `IWSSessionManager.java` and `IWSHttpSession.java` are provided, so you can use them as the starting point for defining your own session managers and session objects.

`IWSSessionManager` extends `IWSHttpSessionManager`. The class file for `IWSHttpSessionManager` is in the JAR file `webserv-rt.jar` in the directory *install_dir*/`lib`. The `IWS60` implements all of the methods in `IWSHttpSessionManager` that need to be implemented, so you can use `IWSSessionManager` as an example of how to extend `IWSHttpSessionManager`. When compiling your subclass of `IWSSessionManager` or `IWSHttpSessionManager`, be sure that the JAR file `webserv-rt.jar` is in your compiler's classpath.

The `JdbcStore.java` and `FileStore.java` source files and the source file for the parent class, `SessionDataStore.java`, are provided so you can modify the session persistence mechanism of `IWS60`. These files are also located in the *install-dir*/`lib` directory.

Note – The `session-timeout` parameter specified in `web.xml` will not be effective when `IWS60` is used.

`MMap` **(UNIX Only)**

MMap is a persistent memory map (`mmap`), file-based session manager that works in both single-process and multi-process mode.

Note – The `MaxProcs` directive in the `magnus.conf` file determines whether the server is running in single-process or multi-process mode. The Maxprocs is deprecated in Web Server 7.0.

Enabling `MMap`

You can enable `MMap` to change its default parameters. You can also enable `MMap` for a particular context if the server is running in single-process mode. To do so, edit the `sun-web.xml` file for the web application as in the following example. Note that `persistence-type` must be set to `mmap`.

```
<sun-web-app>
        ...
        <session-config>
            <session-manager persistence-type="mmap">
              ...
          </session-manager>
           ...
        </session-config>
        ...
</sun-web-app>
```

For more information about the `sun-web.xml` file, see Chapter 9.

Manager Properties for `MMap`

The following table describes `manager-properties` properties for the `MMap` session manager. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE 6–7 `manager-properties` Properties for `MMap`

Property Name	Default Value	Description
`maxSessions`	`1000`	The maximum number of sessions maintained by the session manager at any given time. The session manager refuses to create any more new sessions if there are already `maxSessions` number of sessions present at that time.
`maxValuesPerSession`	`10`	The maximum number of values or objects a session can hold.
`maxValueSize`	`4096`	The maximum size of each value or object that can be stored in the session.
`timeOut`	`1800`	The amount of time in seconds after a session is last accessed by the client before the session manager destroys it. Those sessions that haven't been accessed for at least `timeOut` seconds are destroyed by the reaper method. The `session-timeout` parameter specified in `web.xml` will not be effective when `IWS60` is used.
`reapInterval`	`600`	The amount of time in seconds that the `SessionReaper` thread sleeps before calling the reaper method again.
`maxLocks`	`1`	The number of cross-process locks to use for synchronizing access to individual sessions across processes. The default value is used if the value `0` is specified. This parameter is ignored in single-process mode.

Note – `MMap` can only store objects that implement `java.io.Serializable`.

CHAPTER 7

Developing Lifecycle Listeners

This chapter provides a basic overview, and a description of various features of lifecycle listeners in Sun Java System Web Server 7.0. It includes the following sections:

Server Lifecycle Events

Sun Java System Web Server 7.0 goes through different events in its lifecycle:

- `init`: This event includes reading configuration, initializing built-in subsystems, naming, security and logging services, and creating the web container.
- `startup`: loading and initializing deployed applications.
- `service`: The server is ready to service requests shutdown: stopping and destroying loaded applications. The system is preparing to shutdown.
- `terminating`: The container is being closed. Terminating the built-in subsystems and server runtime environment.
- `reconfig`: A transient server state in which a server thread is dynamically reconfiguring (while rest of server in the service state). This can occur several times during the life of the server.

The LifecycleListener Interface

Sun Java System Web Server 7.0 allows you to write classes and customize various phases of the server lifecycle. For instance, you may have a startup code that ensures a remote data source is available for the applications. Such classes are notified by server lifecycle events. For this purpose, Sun Java System Web Server 7.0 defines a `LifecycleListener` interface that users can implement and register with the Server.

Syntax: `public void handleEvent(LifecycleEvent event)`:receives a lifecycle event.

In its `event` parameter, the programmatic interface for LifecycleListener provides the following to the implementation classes:

- Access to initialization parameters.
- A handler to the server run time environment for naming, logging and accessing resources.
- Exception handling mechanics.

The LifecycleEvent Class

The `LifecycleEvent`class (it is an interface from the point of view to the developer, even if programmatically it is a class) is the means by which these events are represented. This this class informs you of the kind of event that happened (by the `getEventType()` method) and the data associated with the event (by the `getData()` method).

The Server Lifecycle Event Context

The `LifecycleEventContext` interface provides an access to the server runtime environment including the JNDI naming context and logging service. The following methods are defined in this interface.

- `public String[] getCmdLineArgs()`: returns the server command line arguments.
- `public javax.naming.Context getNamingContext()`: returns the naming context.
- `public String getInstallRoot()`: returns the installation root.
- `public String getInstanceName()`: returns the server instance name.
- `public void log(java.util.logging.Level level, String message)`: logs the message to the server log, with verbosity level.
- `public void log(java.util.logging.Level level, String message, Throwable throwable)`: logs the message and the stack trace for throwable, with verbosity level.

The following two methods are also exposed by this interface to keep backward compatibility with the 6.1 version of Web Server:

- `public void log(String message, Throwable throwable)`:logs the message and the stack trace for throwable, with verbosity level.
- `public javax.naming.Context getInitialContext()` : Similar to `getNamingContext()`

Deploying a Lifecycle Module

Server lifecycle listener classes are visible in the serve applications management area. You can add, delete, update, enable, and disable listener classes and set their parameters. Sun Java System Web Server 7.0 will not support dynamic deployment of startup and shutdown classes. Any changes to these classes or their configuration requires server restart.

TABLE 7–1 Elements of the lifecycle

Configurable element / attribute	Data type and Units	Range of values	Remarks
`lifecycle-module.name`	String	Any non-null/non-empty unique string (in lifecycle modules).	Must be specified while registering this lifecycle module.
`lifecycle-module.class`	String	Fully qualified Java class name.	Must implement LifecycleListener interface.
`lifecycle-module.enabled`	Boolean	`true` or `false`.	default 'true'.
`lifecycle-module.load-order`	Integer	0-100 Reserved. 100-MAXINT.	Order of loading the lifecycle event listeners, in the numerical order. Choose a load-order greater than or equal to 100 to avoid conflicts with internal lifecycle modules.
`lifecycle-module.is-failure-fatal`	Boolean	`true` or `false`	If you want server to treat exceptions thrown from the listener classes as fatal, and prevent continuation of normal startup, set this element to true.

TABLE 7–1 Elements of the lifecycle *(Continued)*

Configurable element / attribute	Data type and Units	Range of values	Remarks
`lifecycle-module.class-path`	String	Optional	Points to the user-specified classpath for the listener class.
`lifecycle-module.description`	Element	Optional	Describes lifecycle module.
`property.name`	String	Optional	User-specified parameter `name`. Part of the property element.
`property.value`	String.	Optional	User-specified parameter value. Part of the property element.
`property.description`	String	Optional	User-specified description. Part of the property element.

Considerations for Lifecycle Modules

Following are the considerations of lifecycle module:

- The server lifecycle listener classes are called synchronously from the main server thread and hence extra precaution must be taken to ensure that the listener classes don't block the server.
- The listener classes may create threads if appropriate. The threads must be stopped during the shutdown/termination phases.
- The resources allocated during initialization or startup events should be cleared.
- The listener classes are loaded in the context of server's root class loader, which loads server-wide resources as well. Hence, all the support classes needed by these server lifecycle event listener must be available at this class loader or its parent, system class loader. As a consequence, you must ensure that the Java security manager policy files are appropriately set up; otherwise, a lifecycle listener class trying to perform a `System.exec()` may get a security access violation.

Sample Configuration

The following example shows a portion of the `server.xml` that defines a lifecycle listener.

```
<lifecycle-module>
<class-name>com.sun.ias.server.LifecycleListenerImpl</class-name>
<is-failure-fatal>false</is-failure-fatal>
<description>Sample lifecycle module</description>
<property>
<name>foo</name>
<value>fooval</value>
<property>
</lifecycle-module>
```

The following example shows a sample LifecycleListener implementation

```
/**
*PROPERITARY/CONFIDENTIAL. Use of this product is subject to license terms
*
*Copyright 2006-2007 by SunMicrosystems, Inc.,
*901 San Antonio Road, Palo Alto, California, 94303, U.S.A
*All rights reserved.
package com.sun.ias.server;
import java.util.Properties;
import java.util.logging.Level;
import com.sun.appserv.server.LifecycleEventContext;
import com.sun.appserv.server.ServerLifecycleException;
import com.sun.appserv.server.LifecycleEvent;
import com.sun.appserv.server.LifecycleListener;
 /**
  * LifecycleListenerImpl is a dummy implementation for the LifecycleListener
  * interface.
  * This implementation stubs out various lifecycle interface methods.
  */
public class LifecycleListenerImpl implements LifecycleListener {
/** receive a server lifecycle event
* @param event associated event
* @throws <code>ServerLifecycleException</code> for exception condition.
*
* /
public void handleEvent(LifecycleEvent event) throws ServerLifecycleException {
LifecycleEventContex ctx=event.getLifecycleEventContext();

ctx.log(level.INFO, "got event" + event.getEventType() + "event data:" +
event.getData());

Properties props;
if (Lifecycleevent.INIT_EVENT == event.getEventType()) {
System.out.println("LifecycleListener: INIT_EVENT");

props = (Properties) event.getData();

//handle INIT_EVENT
return;
}

if (LifecycleEvent.STARTUP_EVENT == event.getEventType()) {
System.out.println("LifecycleListener: START_EVENT");
//handle STARTUP_EVENT
return;
}
if (LifecycleEvent.READY_EVENT == event.getEventType()) {
System.out.println("LifecycleListener: READY_EVENT");
//handle READY_EVENT
return;
}

if (LifecycleEvent.SHUTDOWN_EVENT == event.getEventType()) {
System.out.println("LifecycleListener: SHUTDOWN_EVENT");
//handle SHUTDOWN_EVENT
return;
} if (LifecycleEven.TERMINATION_EVENT == event.getEventType()) {
System.out.println("LifecycleListener: TERMINATION_EVENT");
//handle TERMINATION_EVENT
return;
 }
 }
}
```

CHAPTER 8

Securing Web Applications

This chapter describes the basic goals and features of Sun Java System Web Server 7.0 security features related to the Java Servlet Container, and describes how to write secure Java web applications containing components that perform user authentication and access authorization tasks. Because it is helpful to have a basic understanding of security responsibilities and terminology, the beginning of the chapter discusses those topics.

This chapter has the following sections:

Supported Security Features

In an enterprise computing environment there are many security risks. The goal of Sun Java System Web Server 7.0 is to provide highly-secure, interoperable, and distributed component computing based on the Java EE security model. The security goals for Sun Java System Web Server 7.0 include the following:

- Full compliance with the Java Servlet 2.4 security model, including role-based authorization. For more information, see the Security chapter in the Java Servlet 2.4 specification, which can be downloaded from:

  `http://java.sun.com/products/servlet/download.html`
- Support for single sign-on across all Sun Java System Web Server 7.0 applications within a single security domain.
- Support for several underlying authentication realms, such as simple file and LDAP. Certificate authentication is also supported for SSL client authentication. For Solaris, OS platform authentication is supported in addition to these.
- Support for declarative security via Sun Java System Web Server 7.0 specific XML-based role mapping.
- Support for Java Security Manager enforcement.

For more information about Java EE-based security, see the Chapter 6, “Certificates and Keys,” in *Sun Java System Web Server 7.0 Administrator's Guide*.

Common Security Terminology

Before getting into the specific security features and capabilities of Sun Java System Web Server 7.0, it is helpful to have a basic understanding of the common security terminology used throughout this chapter. This section provides that overview.

The most common security processes are authentication, authorization, realm assignment, and role mapping. The following sections define this and other common security terminology:

- “Authentication” on page 112
- “Authorization” on page 113
- “Realms” on page 113
- “Java EE Application Role Mapping” on page 113

Authentication

Authentication verifies the user. For example, the user may enter a user name and password in a web browser, and if those credentials match the permanent profile stored in the active realm, the user is authenticated. The user is associated with a security identity for the remainder of the session. For more information on authentication realms, see “Managing Authentication Realms” in *Sun Java System Web Server 7.0 Administrator's Guide*.

Authorization

Authorization permits a user to perform the desired operations, after being authenticated. For example, a human resources application may authorize managers to view personal employee information for all employees, but allow employees to view only their own personal information.

Realms

A realm, also called a security policy domain or a security domain in the Java EE specification, is a scope over which a common security policy is defined and enforced by the security administrator of the security service. Supported realms in Sun Java System Web Server 7.0 are `file`, `ldap`, `certificate`, `solaris`, `custom`, and `native`.

Java EE Application Role Mapping

In the Java EE/Servlet security model, a client may be defined in terms of a security role. For example, a company might use its employee database to generate both a company-wide phone book application and payroll information. Obviously, while all employees might have access to phone numbers and email addresses, only some employees would have access to the salary information. Employees with the right to view or change salaries might be defined as having a special security role.

A role is different from a user group in that a role defines a function in an application, while a group is a set of users who are related in some way. For example, members of the groups astronauts, scientists, and (occasionally) politicians all fit into the role of SpaceShuttlePassenger.

In Sun Java System Web Server 7.0, roles correspond to users or groups (or both) configured in the active realm.

Security Features Specific to the Web Server

In addition to supporting the Java EE 1.4 security model, Sun Java System Web Server 7.0 also supports the following features that are specific to the Web Server:

- Single sign-on across all Sun Java System Web Server 7.0 applications within a single security domain.
- Programmatic login.
- The parallel Access Control List (ACL)-based security model, in addition to the Java EE/Servlet security model.

- Support for secure ACL-based Java web applications, in addition to native content.

This section discusses the following:

- “Web Server Security Model” on page 114
- “Web Application and URL Authorizations” on page 116

Web Server Security Model

Secure applications require a client to be authenticated as a valid application user and have authorization to access servlets and JSPs.

Applications with a secure web container may enforce the following security processes for clients:

- Authenticate the caller.
- Authorize the caller for access to each servlet/JSP based on the applicable access control configuration.

Authentication is the process of confirming an identity. Authorization means granting access to a restricted resource to an identity, and access control mechanisms enforce these restrictions. Authentication and authorization can be enforced by a number of security models and services.

Sun Java System Web Server 7.0 provides authentication and authorization support through the following mechanisms, which are discussed in this section:

- ACL-based authentication and authorization
- Java EE/Servlet-based authentication and authorization

Whether performed by the ACL subsystem or the Java EE/Servlet authentication subsystem, authentication and authorization are still the two fundamental operations that define secure web content.

ACL-Based Authentication and Authorization

ACL-based access control is described at length in the “Configuring Access Control” in *Sun Java System Web Server 7.0 Administrator’s Guide*. This following section provides a brief overview of the key concepts.

Sun Java System Web Server 7.0 supports authentication and authorization through the use of locally stored ACLs, which describe what access rights a user has for a resource. For example, an entry in an ACL can grant a user named John `read` permission to a particular folder named `misc`:

```
acl "path=/export/user/990628.1/docs/misc/";
  authenticate (user,group) {
      database = "default";
      method = "basic";
   };
   deny (all)
  (user = "John");
   allow (read);
```

The core ACLs in Sun Java System Web Server 7.0 support three types of authentication: basic, certificate, and digest.

Basic authentication relies on lists of user names and passwords passed as cleartext. Certificates bind a name to a public key. Digest authentication uses encryption techniques to encrypt the user's credentials.

The main features of the ACL-based access control model are described below:

- ACL-based authentication uses the following configuration files:
  - *install_dir*`/config/*.acl` files
  - *install_dir*`/config/server.xml`

  Authentication is performed by `auth-db` modules that are configured in the `server.xml` file.
- Authorization is performed by access control rules set in the *install_dir*`/config/*.acl` files, if ACLs are configured.

In addition, the Sun Java System Web Server 7.0 SSL engine supports external crypto hardware to offload SSL processing and to provide optional tamper-resistant key storage.

For more information about ACL-based access control and the use of external crypto hardware, see the Sun Java System Web Server 7.0 *Administrator's Guide*.

Java EE/Servlet-Based Authentication and Authorization

Sun Java System Web Server 7.0, apart from providing ACL-based authentication, also implements the security model defined in the Java EE 1.4 specification to provide several features that help you develop and deploy secure Java web applications.

A typical Java EE-based web application consists of the following parts, access to any or all of which can be restricted:

- Servlets
- JavaServer Pages (JSP) components
- HTML documents
- Miscellaneous resources, such as image files and compressed archives

The Java EE/Servlet-based access control infrastructure relies on the use of security realms. When a user tries to access the main page of an application through a web browser, the web container prompts for the users credential information, and then passes it for verification to the realm that is currently active in the security service.

A realm, also called a security policy domain or security domain in the Java EE specification, represents a set of known users along with optional group membership information. The main implementation also encapsulates a mechanism for performing authentication against the data set.

The main features of the Java EE/Servlet-based access control model are described below:

- Java EE/Servlet-based authentication uses the following configuration files:
  - The web application deployment descriptor files `web.xml` and `sun-web.xml`
  - *install_dir*`/config/server.xml`

  Authentication is performed by Java security realms that are configured through `<auth-realm>` entries in the `server.xml` file.
- Authorization is performed by access control rules in the deployment descriptor file, `web.xml`, in case any such rules have been set.

Web Application and URL Authorizations

Secure web applications may have authentication and authorization properties. The web container supports three types of authentication: basic, certificate, and form-based. The core ACLs support basic, certificate, and digest. For more information about ACL configuration, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

When a browser requests an application URL which requires authentication, the web container collects the user authentication information (for example, user name and password) and passes it to the security service for authentication.

For Java EE web applications, Sun Java System Web Server 7.0 consults the application's `web.xml` for information on which parts of the application are protected, and which role(s) are authorized to access. It also consults `sun-web.xml` to see whether the currently authenticated user belongs to one of the required roles, either directly via user mapping or indirectly via group mapping.

Container Security

The component containers are responsible for providing Java EE application security. Two security forms are provided by the container, as discussed below: programmatic security and declarative security.

Programmatic Security

In programmatic security, a servlet uses method calls to the security API, as specified by the Java EE security model, to make business logic decisions based on the caller or remote user's security role. Programmatic security should only be used when declarative security alone is insufficient to meet the application's security model.

The Java EE 1.4 specification defines programmatic security with respect to servlets as consisting of two methods of the servlet `HttpServletRequest` interface. Sun Java System Web Server 7.0supports these interfaces as defined in the specification.

In addition to the programmatic security defined in the Java EE specifications, Sun Java System Web Server 7.0 also supports programmatic login. For more information, see "Programmatic Login" on page 131

Declarative Security

Declarative security means that the security mechanism for an application is declared and handled externally to the application. Deployment descriptors describe the Java EE application's security structure, including security roles, access control, and authentication requirements.

Sun Java System Web Server 7.0 supports the DTDs specified by the Java EE 1.4 specification, and has additional security elements included in its own deployment descriptors. Declarative security is the application deployers responsibility.

User Authentication by Servlets

Web Server supports the web-based login mechanisms required by the Java EE 1.4 specification are supported by the Sun Java System Web Server 7.0. These mechanisms are discussed in this section:

- "HTTP Basic Authentication" on page 118
- "SSL Mutual Authentication" on page 118
- "Form-Based Login" on page 118

The `login-config` element in the `web.xml` deployment descriptor file describes the authentication method used, the application's realm name displayed by the HTTP basic authentication, and the form login mechanism's attributes.

The `login-config` element syntax is as follows:

```
<!ELEMENT login-config (auth-method?,realm-name?,form-login-config?)>
```

Note – The auth-method subelement of login-config is optional, but if it is not included, the server defaults to HTTP Basic Authentication, which is not very secure.

For more information about web.xml elements, see the Java Servlet 2.4 specification. You can find the specification here:

http://java.sun.com/products/servlet/download.html

For more information on sun-web.xml elements, see Chapter 9.

HTTP Basic Authentication

HTTP basic authentication (RFC 2617) is supported by the Sun Java System Web Server 7.0. Because passwords are sent with base64 encoding, this authentication type is not very secure. Use of SSL or another equivalent transport encryption is recommended to protect the password during transmission.

SSL Mutual Authentication

SSL 3.0 and the means to perform mutual (client/server) certificate-based authentication is a Java EE 1.4 specification requirement. This security mechanism provides user authentication using HTTPS (HTTP over SSL). For more information see, “Creating a Configuration” in *Sun Java System Web Server 7.0 Administrator’s Guide*.

Form-Based Login

The login screen's look and feel cannot be controlled with the HTTP browser's built-in mechanisms. Java EE has the ability to package a standard HTML or servlet/JSP based form for logging in. The login form is associated with a web protection domain and is used to authenticate previously unauthenticated users.

Because passwords are sent (unless protected by the underlying transport), this authentication type is not very secure. Use of SSL or another equivalent transport encryption is recommended to protect the password during transmission.

For the authentication to proceed appropriately, the login form action must always be j_security_check. For more information see, Chapter 4.

The following is an HTML sample showing how to program the form in an HTML page:

```
<form method="POST" action="j_security_check">
        <input type="text" name="j_username">
         <input type="password" name="j_password">
```

```
    </form>
```

You can specify the parameter encoding for the form. For details, see "parameter-encoding" on page 197.

User Authentication for Single Sign-On

Single sign-on across applications on the Sun Java System Web Server 7.0 is supported by the Sun Java System Web Server 7.0 servlets and JSPs. This feature allows multiple applications that require the same user sign-on information to share this information between them, rather than having the user sign on separately for each application. These applications are created to authenticate the user one time, and when needed this authentication information is propagated to all other involved applications.

An example application using the single sign-on scenario could be a consolidated airline booking service that searches all airlines and provides links to different airline web sites. Once the user signs on to the consolidated booking service, the user information can be used by each individual airline site without requiring another sign on.

Single sign-on operates according to the following rules:

- Single sign-on applies to web applications configured for the same realm and virtual server. The realm is defined by the `realm-name` element in the `web.xml` file. For information about virtual servers, see the *Sun Java System Web Server 7.0 Administrator's Guide*.
- As long as users access only unprotected resources in any of the web applications on a virtual server, they are not challenged to authenticate themselves.
- As soon as a user accesses a protected resource in any web application associated with a virtual server, the user is challenged to authenticate him or herself, using the login method defined for the web application currently being accessed.
- Once authenticated, the roles associated with this user are used for access control decisions across all associated web applications, without challenging the user to authenticate to each application individually.
- When the user logs out of one web application (for example, by invalidating or timing out the corresponding session if form-based login is used), the user's sessions in all web applications are invalidated. Any subsequent attempt to access a protected resource in any application requires the user's authorization.
- The single sign-on feature utilizes HTTP cookies to transmit a token that associates each request with the saved user identity, so it can only be used in client environments that support cookies.

To configure single sign-on, set the following properties in the `single-sign-on` element of the `server.xml` file:

- `enabled`: If `false`, single sign-on is disabled for this virtual server, and users must authenticate separately to every application on the virtual server. The default is `false`.
- `idle-timeout`: Specifies the time after which a user's single sign-on record becomes eligible for purging if no client activity is received. Since single sign-on applies across several applications on the same virtual server, access to any of the applications keeps the single sign-on record active. The default value is 5 minutes (300 seconds). Higher values provide longer single sign-on persistence for the users at the expense of more memory use on the server.

  To configure single sign-on through CLI, see the `enable-single-signon`(1) and `disable-single-signon`(1)

Here is an example configuration with all default values:

```
<single-sign-on>
        <enabled>1</enabled>
         <idle-timeout>300</idle-timeout>
      </single-sign-on>
```

User Authorization by Servlets

Servlets can be configured to only permit access to users with the appropriate authorization level. This section covers the following topics:

- "Defining Roles" on page 120
- "Defining Servlet Authorization Constraints" on page 121

Defining Roles

Security roles define an application function, made up of a number of users, groups, or both. The relationship between users and groups is determined by the specific realm implementation being used.

You can define roles in the Java EE deployment descriptor file, `web.xml`, and the corresponding role mappings in the Sun Java System Web Server 7.0 deployment descriptor file, `sun-web.xml`. For more information about `sun-web.xml`, see Chapter 9.

Each `security-role-mapping` element in the `sun-web.xml` file maps a role name permitted by the web application to principals and groups. For example, a `sun-web.xml` file for a deployed web application might contain the following:

```
     <sun-web-app>
        <security-role-mapping>
           <role-name>manager</role-name>
```

```
            <principal-name>jgarcia</principal-name>
            <principal-name>mwebster</principal-name>
            <group-name>team-leads</group-name>
        </security-role-mapping>
        <security-role-mapping>
            <role-name>administrator</role-name>
            <principal-name>dsmith</principal-name>
        </security-role-mapping>
    </sun-web-app>
```

Note that the `role-name` in this example must match the `role-name` in the `security-role` element of the corresponding `web.xml` file.

For web applications, the roles are always specified in the `sun-web.xml` file. A role can be mapped to either specific principals or to groups (or both). The principal or group names used must be valid principals or groups in the current realm.

Defining Servlet Authorization Constraints

On the servlet level, you can define access permissions using the `auth-constraint` element of the `web.xml` file.

The `auth-constraint` element on the resource collection must be used to indicate the user roles permitted to the resource collection. Refer to the Java Servlet specification for details on configuring servlet authorization constraints.

Fetching the Client Certificate

When you enable SSL and require client certificate authorization, your servlets have access to the client certificate as shown in the following example:

```
if (request.isSecure()) {
         java.security.cert.X509Certificate[] certs;
         certs = request.getAttribute("javax.servlet.request.X509Certificate");
        if (certs != null) {
            clientCert = certs[0];
            if (clientCert != null) {
               // Get the Distinguised Name for the user.
               java.security.Principal userDN = clientCert.getSubjectDN();
               ...
            }
         }
      }
```

The `userDn` is the fully qualified distinguished name for the user.

Using Realms

This section provides an overview of the configuration characteristics of the supported realms. For detailed information about configuring realms, see the "Managing Authentication Realms" in *Sun Java System Web Server 7.0 Administrator's Guide*.

Using Web Services Message Security

In message security, security information is inserted into messages that travels through the networking layers and reaches at the message destination(s).

- Setting up `wadm`
- Configuring Message Security Provider
- Message Security Provider in Application

Configuring the Web Server for Message Security

This section contains the following topics:

- Actions of Request and Response Policy Configurations
- To configure other security facilities
- Security Enhancements to `server.xml`
- Security Enhancements to `sun-web.xml`

Actions of Request and Response Policy Configurations

The following table shows message protection policy configuration and the resulting message security operations performed by the WS-Security SOAP message security providers for that configuration.

TABLE 8–1 Message Protection Policy Configuration

Message protection policy	Resulting WS-Security SOAP message protection operations
`auth-source= "sender"`	The message contains `wase:security` header that contains a `wsse:UsernameToken` (with password).

TABLE 8–1 Message Protection Policy Configuration *(Continued)*

`auth-source="content"`	The content of the SOAP message body is signed. The message contains a `wsse:Security` header that contains the message Body signature represented as a `ds:Signature`.
`auth-source="sender"` `auth-recipient="before-content" OR` `auth-recipient="after-content"`	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The message contains a `wsse:Security` header that contains a `wsse:UsernameToken` (with password) and an `xenc:EncryptedKey`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
`auth-source= "content"` `auth-recipient= "before-content"`	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The `xenc:EncryptedData` is signed. The message contains a `wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds:Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
`auth-source="content"` `auth-recipient="after-content"`	The content of the SOAP message Body is signed, then encrypted, and then replaced with the resulting `xend:EncryptedData`. The message contains a `wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds:Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
`auth-recipient="before-content"` `OR` `auth-recipient="after-content"`	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The message contains a `wsse:Security` header that contains an `xenc:EncryptedKey`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
No policy specified.	No security operations are performed by the modules.

▼ To Configure Other Security Facilities

The Web Server implements message security using message security providers integrated in its SOAP processing layer. The message security providers depend on other security facilities of Web Server.

Steps

1 **If using a username token, configure a user database, if necessary.**

When using a username/password token, an appropriate realm must be configured and an appropriate user database must be configured for the realm.

2 **Manage certificates and private keys, if necessary.**

After configuring the Web Server facilities for use by message security providers as described in, "Managing Certificates" in *Sun Java System Web Server 7.0 Administrator's Guide*.

Security Enhancements to server.xml

The `server` element in `server.xml` is extended to contain one or more `soap-auth-provider` elements, each of which contains a list of configured soap message security providers. The `server` element also includes a `default-soap-auth-provider-name` for the default SOAP message-level authentication provider.

See Chapter 3, "Elements in server.xml," in *Sun Java System Web Server 7.0 Administrator's Configuration File Reference* for more information.

Administration Command-Line Interface (CLI) is provided to add, remove, and list the `soap-auth-provider` element in `server.xml`. The CLI also supports adding a `deafult-soap-auth-provider-name` to `server.xml`.

Security Enhancements to sun-web.xml

Security-related additions to `sun-web.xml` are described in detail in the following tables.

`webservice-endpoint` Element

```
<!ELEMENT webservice-endpoint (port-component name, endpoint-address-uri?,
(login-config|message-security-binding)?,transport-guarantee?,
service-gname?,tie-class?, servlet-imp-class?)>
```

TABLE 8–2 `webservice-endpoint` Element

Element Name	Occurrences	Description	Type
`port-component-name`	1	Unique name of a Web Service within a module. This name should be the same as the endpoint::name in `sun-jaxws.xml`.	PCDATA

TABLE 8–2 webservice-endpoint Element (Continued)

`endpoint-address-uri`	0 or 1	Unused for Web Server	PCDATA
`login-config`		Unused for Web Server	
`message-security-binding`	0 or 1	Used to bind a Web Service endpoint or port to a specific security provider. This element can also be used to optionally provide a definition of message security requirements to be enforced by the security provider.	See Table 8–3 `message-security–binding`
`transport-guarantee`	0 or 1	Unused for Web Server	PCDATA
`service-qname`	0 or 1	Unused for Web Server	
`tie-class`	0 or 1	Unused for Web Server	PCDATA
`servlet-impl-class`	0 or 1	Unused for Web Server	Class name

`message-security-binding` Element

The `message-secuirty-binding` element is used to bind a web service endpoint or port to a specific security provider.

Syntax:

```
<!ELEMENT message-security-binding (message-secuirty*)>
<!ATLIST message-secuirty-binding
auth-layer %message-layer;#REQUIRED
provider-id CDATA #IMPLIED >
```

TABLE 8–3 `message-secuirty-binding` Element

Element name	Occurrences	Description	Type
`message-security`	0 or more	Specifies the message security requirements of request and response for the endpoint or port	Table 8–5

TABLE 8–4 Attributes of `message-security-binding` Element

Attribute name	Description	Type	Default
`auth-layer`	Layer at which the security should be enforced	Entity message-layer	This attribute is required.
provider-id	Identifies the provider-config that should be used	CDATA	If a value is not specified, then the default provider is used. If there is no default provider at the layer, the authentication requirements defined in the message-security-binding are not enforced.

`message-security` Element

The syntax for `message-security` element:

```
<!ELEMENT message-security (message+, request-protection?, response-protection?
)>
```

TABLE 8–5 `message-security` element

Element name	Occurrences	Description	Type
`message`	1 or more	Describes the methods or operations to which the security requirements apply	Table 8–6
`request-protection`	0 or 1	Describes the authentication requirements applicable to a request	Table 8–7
`response-protection`	0 or 1	Describes the authentication requirements applicable to a response	Table 8–8

`message` Element

The syntax for `message` element: `<!ELEMENT (java-method?|operatio-name?)>`.

TABLE 8–6 `message` element

Element name	occurrences	Description	Type
`java-method`	0 or 1	Java methods on which the security should be enforced	Table 8–9
`operation-name`	0 or 1	WSDL name of an operation of the web service	PCDATA

Attributes of `request-protect` Element

The syntax for `request-protect` element is as follows.

```
<!ELEMENT request-protection EMPTY>
<!ATTLIST request-protection
auth-source (sender|content)#IMPLIED
auth-recipient (before-content |after-content)#IMPLIED
```

TABLE 8–7 `request-protection` element

Attribute name	Description	Value	Default
`auth-source`	Defines a requirement for message layer sender authentication (for example, username password) or content authentication (For example, digital signature)	sender or content	Implied

TABLE 8–7 request-protection element *(Continued)*

auth-recipient	Defines a requirement for message layer authentication of the receiver of a message to its sender (for example, by XML encryption). A before-content attribute value indicates that recipient authentication (for example, encryption) occurs before any content authentication (for example, encrypt then sign).	before-content or after-content	Implied

response-protection Element

The syntax for response-protection element is as follows:

```
<<!ELEMENT response-protection EMPTY>
<!ATTLIST response-protection
auth-source (sender|content)#IMPLIED
auth-recipient (before-content |after-content)#IMPLIED
```

TABLE 8–8 Attributes of the response-protection Element

Attribute name	Description	Value	Default
auth-source	Defines a requirement for message layer sender authentication (for example username password) or content authentication (For example digital signature)	sender or content	Implied

TABLE 8–8 Attributes of the `response-protection` Element *(Continued)*

`auth-recipient`	Defines a requirement for message layer authentication of the receiver of a message to its sender (For example by XML encryption). The before-content attribute value indicates that recipient authentication (For example, encryption) is to occur before any content authentication (For example, encrypt then sign) with respect to the target of the containing auth-policy.	`before-content` or `after-content`	Implied

`java-method` Element

The syntax for `java-method` is as follows:

```
<!ELEMENT java-method (method-name,method-params?)>
```

TABLE 8–9 `java-method` Element

Element name	Occurrences	Description	Value
`method-name`	1	Name of the service method	PCDATA
`method-params`	0 or 1	List of the fully-qualified Java type names of the method parameters.	Table 8–10

`method-params` Element

The syntax for `method-params (method-param*)` element.

TABLE 8–10 Attributes of `method-params` Element

Element name	Occurrences	Description	Value
`method-params`	0 or more	Fully qualified Java type name of a method parameter	PCDATA

message-layer

The `message-layer` entity defines the value of the value of the auth-layer attribute.

Syntax for `message-layer` entity:`<!Entity %message-layer "(SOAP)"`

Using Message Security Provider in an Application

The following `sub-web.xml` example shows how to use the `server.xml`message security provider `provider1` in a web application:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
        Copyright 2006-2007 Sun Microsystems,Inc. All rights reserved.
        Use is subject to license terms.
-->

<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.
//DTD Application Server 8.1 Servlet 2.4//EN"
"http://www.sun.com/software/appserver/dtds/sun-web-app_2_4-1.dtd">
<sun-web-app>
        <context-root>/jaxws-fromwsdl-soap12</context-root>
        <servlet>
            <servlet-name>fromwsdl</servlet-name>
            <webservices-endpoint>
                <port-component-name>fromwsdl-soap12</port-component-name>
                <message-security-binding auth-layer="SOAP"
                <provider-id>provider1</provider-id>
                </message-security-binding>
                </webservices-endpoint>
                </servlet>
        </sun-web-app>
```

Note – The `port-component-name` element should be the same as the `name` attribute in the `endpoint` element in `sun-jaxws.xml`. If the `provider-id` element is not specified in `sun-web.xml`, then the `default-soap-auth-provider-name` configured in `server.xml` will be used as the provider.

Deploy, the sample web application `fromwsdl-soap12.war` on to the Sun Java System Web Server 7.0.

Programmatic Login

Programmatic login allows a deployed Java EE application to invoke a login method. If the login is successful, a `SecurityContext` is established as if the client had authenticated using any of the conventional Java EE mechanisms.

Programmatic login is useful for application with unique needs that cannot be accommodated by any of the Java EE standard authentication mechanisms.

This section discusses the following topics:

- "Precautions" on page 131
- "Granting Programmatic Login Permission" on page 131
- "The ProgrammaticLogin Class" on page 132

Precautions

The Sun Java System Web Server 7.0 is not involved in how the login information (user name and password) is obtained by the deployed application. Programmatic login places the burden on the application developer with respect to assuring that the resulting system meets security requirements. If the application code reads the authentication information across the network, it is up to the application to determine whether to trust the user.

Programmatic login allows the application developer to bypass the Web Server-supported authentication mechanisms and feed authentication data directly to the security service. While flexible, this capability should not be used without some understanding of security issues.

Since this mechanism bypasses the container-managed authentication process and sequence, the application developer must be very careful in making sure that authentication is established before accessing any restricted resources or methods. It is also the application developer's responsibility to verify the status of the login attempt and to alter the behavior of the application accordingly.

The programmatic login state does not necessarily persist in sessions or participate in single sign-on.

Lazy authentication is not supported for programmatic login. If an access check is reached and the deployed application has not properly authenticated using the programmatic login method, access is denied immediately and the application may fail if not properly coded to account for this occurrence.

Granting Programmatic Login Permission

The `ProgrammaticLoginPermission` permission is required to invoke the programmatic login mechanism for an application. This permission is not granted by default to deployed applications because this is not a standard Java EE mechanism.

To grant the required permission to the application, add the following to the *instance_dir*/`config/server.policy` file:

```
grant codeBase "file:jar_file_path" {
    permission com.sun.appserv.security.ProgrammaticLoginPermission
    "login";
 };
```

The `jar_file_path` is the path to the application's JAR file.

Note – If the Security Manager is disabled, it is not mandatory to grant permission.

For more information about the `server.policy` file, see "The server.policy File" on page 133.

The ProgrammaticLogin Class

The `com.sun.appserv.security.ProgrammaticLogin` class enables a user to perform login programmatically.

The login method for servlets or JSPs has the following signature:

```
public Boolean login(String user, String password,
javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
```

This method:

- Performs the authentication
- Returns `true` if the login succeeded, `false` if the login failed

Enabling the Java Security Manager

Sun Java System Web Server 7.0 supports the Java Security Manager. The Java Security Manager is disabled by default when you install the product, which may improve performance significantly for some types of applications. Enabling the Java Security Manager may improve security by restricting the rights granted to your Java EE web applications. To enable the Java Security Manager, add the following JVM options in the server.xml.

```
<jvm-options>-Djava.security.manager</jvm-options>

<jvm-options>-Djava.security.policy=instance_dir

/config/server.policy</jvm-options>
```

where *instance_dir* is the path to the installation directory of this server instance.

Based on your application and deployment needs, you should evaluate whether to run with or without the Security Manager.

Running with the Security Manager on will help catch some spec-compliance issues with Java EE applications. All Java EE applications should be able to run with the Security Manager active and with only the default permissions. For this reason it is recommended that the Security Manager be turned on during development. This will help produce applications that can easily be deployed in environments where the Security Manager is always active (such as some versions of Sun Java System Application Server). Running with the Security Manager also helps isolate applications and may catch inappropriate operations.

The main drawback of running with the Security Manager is that it negatively impacts performance. Depending on the application details and the deployment environment, this impact may be negligible or quite significant.

The server.policy File

Each Sun Java System Web Server 7.0 instance has its own standard Java Platform, Standard Edition (Java SE™ platform) policy file, located in the *instance_dir*/`config` directory. The file is named `server.policy`.

Sun Java System Web Server 7.0 is a Java EE 1.4-compliant web server. As such, it follows the recommendations and requirements of the Java EE specification, including the optional presence of the Security Manager (the Java component that enforces the policy), and a limited permission set for Java EE application code.

This section includes the following topics:

- "Default Permissions" on page 133
- "Changing Permissions for an Application" on page 134

Default Permissions

Internal server code is granted all permissions. These are covered by the `AllPermission` grant blocks to various parts of the server infrastructure code. Do not modify these entries.

Application permissions are granted in the default grant block. These permissions apply to all code not part of the internal server code listed previously.

A few permissions above the minimal set are also granted in the default `server.policy` file. These are necessary due to various internal dependencies of the server implementation. Java EE application developers must not rely on these additional permissions.

Changing Permissions for an Application

The default policy for each instance limits the permissions of Java EE-deployed applications to the minimal set of permissions required for these applications to operate correctly. If you develop applications that require more than this default set of permissions, you can edit the `server.policy` file to add the custom permissions that your applications need.

You should add the extra permissions only to the applications that require them, not to all applications deployed to a server instance. Do not add extra permissions to the default set (the grant block with no `codebase`, which applies to all code). Instead, add a new grant block with a codebase specific to the application requiring the extra permissions, and only add the minimally necessary permissions in that block.

Note – Do not add `java.security.AllPermission` to the `server.policy` file for application code. Doing so completely defeats the purpose of the Security Manager, yet you still get the performance overhead associated with it.

As noted in the Java EE specification, an application should provide documentation of the additional permissions it needs. If an application requires extra permissions but does not document the set it needs, contact the application author for details.

As a last resort, you can iteratively determine the permission set an application needs by observing `AccessControlException` occurrences in the server log. If this is not sufficient, you can add the `-Djava.security.debug=all` JVM option to the server instance. For details, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

You can use the Java SE standard policytool or any text editor to edit the `server.policy` file. For more information, see:

`http://java.sun.com/docs/books/tutorial/security1.2/tour2/index.html`

For detailed information about the permissions you can set in the `server.policy` file, see:

`http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html`

The Javadoc for the `Permission` class is here:

`http://java.sun.com/j2se/1.4.2/docs/api/java/security/Permission.html`

Related Information

The following table describes where you can find more information about security and security configuration topics in the Sun Java System Web Server 7.0 documentation:

TABLE 8–11 More Information on Security-related Issues

For Information On	See
Configuring Java security and realm-based authentication	The chapter "Securing Your Web Server" in the *Sun Java System Web Server 7.0 Administrator's Guide*.
Certificates and public key cryptography	The chapter "Using Certificates and Keys" in the *Sun Java System Web Server 7.0 Administrator's Guide*.
ACL-based security	The chapter "Controlling Access to Your Server" in the *Sun Java System Web Server 7.0 Administrator's Guide*.
Configuring `authentication services in the server.xml` files	The chapter "Controlling Access to Your Server" in the *Sun Java System Web Server 7.0 Administrator's Guide* and *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

CHAPTER 9

Deploying Web Applications

This chapter describes how web applications are assembled and deployed in Sun Java System Web Server 7.0, and has the following sections:

Web Application Structure

Web applications have a directory structure, which is fully accessible from a mapping to the application's document root (for example, `/hello`). The document root contains JSP files, HTML files, and static files such as image files.

A WAR file (Web ARchive file) contains a complete web application in compressed form.

A special directory under the document root, `WEB-INF`, contains everything related to the application that is not in the public document tree of the application. No file contained in `WEB-INF` can be served directly to the client. The contents of `WEB-INF` include:

- `/WEB-INF/classes/*`: The directory for servlet and other classes.
- `/WEB-INF/web.xml` and `/WEB-INF/sun-web.xml`: XML-based deployment descriptors that specify the web application configuration, including mappings, initialization parameters, and security constraints.

The web application directory structure follows the structure outlined in the Java EE specification. The following is an example directory structure of a simple web application:

```
+ hello/
       |--- index.jsp
       |--+ META-INF/
       |  |--- MANIFEST.MF
       --+ WEB-INF/
          |--- web.xml
          --- sun-web.xml
```

Deployment Tools

Using Sun Java Studio Enterprise 8.1

Sun Java System Web Server 7.0 supports Sun Java Studio Enterprise 8.1, Standard Edition. You can use Sun Java Studio to assemble and deploy web applications.

Sun Java Studio technology is Sun's powerful, extensible, integrated development environment (IDE) for Java technology developers. Sun Java Studio Enterprise 8.1 is based on NetBeans™ software, and integrated with the Sun Java platform. (Sun Java Studio Enterprise 8.1 also supports NetBeans 5.0.)

Sun Java Studio support is available on all platforms supported by Sun Java Studio Enterprise 8.1. The plug-in for the Web Server is obtained in the following ways:

- From the Companion CD in the Sun Java Studio Enterprise 8.1 Media Kit
- By using the AutoUpdate feature of Sun Java Studio
- From the download center for Sun Java System Web Server 7.0 at `http://www.sun.com/software/download/index.jsp`

Note – The Sun Java Studio Enterprise 8.1 for Sun Java System Web Server 7.0 works only with a local Web Server (that is, with the IDE and the Web Server on the same machine).

For information about using the web application features in Sun Java Studio Enterprise 8.1, explore the resources at `http://developers.sun.com/prodtech/javatools/jsstandard/reference/docs/index.html`.

The behavior of the Sun Java Studio Enterprise 8.1 plug-in for Sun Java System Web Server 7.0 is the same as that for Sun™ Java System Application Server 8. If you're using the Web Application Tutorial at the site listed above, for instance, you would set the Sun Java System Web Server 7.0 instance as the default, and then take the same actions described in the tutorial.

For more information about Sun Java Studio Enterprise 8.1, visit `http://www.sun.com/software/sundev/jde/`

Note – For basic information about using Sun Java Studio Enterprise 8.1 to debug web applications, see "Using Developer tools for Debugging" on page 153.

Using NetBeans IDE 5.0

NetBeans IDE 5.0 is an integrated development environment to create, deploy, and Java EE-based web applications. This section describes how to use the NetBeans IDE 5.0 to create and deploy the web applications for Sun Java System Web Server 7.0.

▼ To Install NetBeans IDE 5.0

1. **Download and install NetBeans IDE 5.0 from `http://webserver.netbeans.org`.**
2. **After installation, launch the NetBeans IDE 5.0.**
3. **Download the latest plug-in.**
4. **Extract the *org-netbeans-modules-j2ee-sun-ws7.nbm* file.**
5. **Select Update Center, from the Tools menu.**
6. **Select Install Manually Downloaded Modules (.nbm Files) and click Next.**
7. **Add the downloaded .nbm files and click Next.**
8. **Select Sun Java System Web Server 7.0 and click Next.**
9. **Check the Include box and click Yes to install the plug-in.**

   Web server 7.0 plug-in is installed in the IDE.

▼ To Register Web server 7.0 in the NetBeans IDE 5.0

1. **Select Server node in the Runtime Tab, right click and choose Add Server.**
2. **Select the Sun Java System Web Server 7.0 to register the Web Server and click Next.**
3. **Fill the details of the Sun Java System Web Server 7.0 installed on the local system or remote machine.**

Note – The plug-in requires a local Web Server installation on the same machine, even if you are registering a remote Web server. The local installation is required for some of the Web Server jar files.

---

4 **An additional step is required if the Web Server contains multiple configurations. You only need to do it once. From the IDE drop-down list, select the Web Server nod. Choose one configuration to use for deployment from the IDE.**

Note – The drop down list contains configuration which have at least one virtual server and one instance.

---

5 **Once the server is registered, you can see it in the Server node of Runtime tab. Right click and choose Start.**

6 **You can expand the nodes and see all the web applications and resources.**

▼ Deploying Web Applications

1 **Select New Project from the File menu.**

2 **In the category list select Web, and in projects select web application. click Next.**

3 **From the Server drop-down, select Sun Java System Web Server 7.0.**

4 **Click Next and complete the web project creation.**

Once the project is created, web project is displayed in the Projects tab. You can find all basic files created in the Web Server specific deployment descriptor `sun-web.xml`.

The web application is now ready to compile, deploy.

Creating Web Deployment Descriptors

Sun Java System Web Server 7.0 web applications include two deployment descriptor files:

- A Java EE standard file (`web.xml`), described in the Java Servlet 2.4 specification. You can find the specification here: `http://java.sun.com/products/servlet/download.html`
- An optional Sun Java System Web Server 7.0 -specific file (`sun-web.xml`), described later in this chapter.

The easiest way to create the `web.xml` and `sun-web.xml` files is to deploy a web application using Sun Java Studio Enterprise 8.1. For sample `web.xml` and `sun-web.xml` files, see "Sample Web Application XML Files" on page 203

Deploying Web Applications

You can deploy a web application using either the Admin console or the command-line interface.

▼ To Deploy Using Admin Console

Before You Begin Select the virtual server, in which you need to deploy the web application.

1. **Click the Add Web Application tab in the home page.**

   The Add Web Application screen appears.

2. **Specify the location or package file path to upload to the Web Server.**

3. **Enter the URI.**

   Specify the URI for your web application. This will be the application's context root and is relative to the server host.

4. **Select the check box to enable JSP pre-compilation.**

5. **Click OK.**

   The Web Application page appears.

6. **Click Save.**

7. **Click Deployment Pending link in the top right of the screen.**

   The Configuration Deployment screen displays.

8. **Click Deploy.**

   The web application is deployed.

   For more information about using the Administration, see the *Sun Java System Web Server 7.0 Administrator's Guide*

Deploying Using wadm

Note – Before you can manually deploy a web application, make sure that the *server_root*`/bin` directory is in your path.

You can use the wadm utility at the command line to deploy a WAR file into a virtual server web application environment as follows:

```
wadm [--user=admin-user] [--password-file=admin-pswd-file]
[--host=admin-host]
[--port=admin-port][--no-ssl]
[--rcfile=rcfile][--no-prompt][--commands-file=]filename
```

For more information on how to add, enable, and disable web applications, see `add-webapp`(1)

The following table describes the command parameters. The left column lists the parameter, and the right column describes the parameter.

TABLE 9–1 Command Parameters

Parameter	Description
`--user`	Specify the user name of the authorized Web Server administrator.
`--pasword-file`	Specify the password file. The password file contains the password to authenticate administrators to the administration server. This file must contain a line `wadm_password=password`. If you do not specify this option, you will be prompted for a password while executing this command.
`--host`	Specify the name of the machine where the administration server is running. The default host is `localhost`.
`--port`	Specify the port number of the administration server. The default non-SSL port is 8800 and the default SSL port is 8989.
`--no-ssl`	Specify this option to use plain text connection to communicate with the administration server. The default connection is SSL.
`--rcfile`	Specify the name of the `rcfile` that has to be loaded while starting the `wadm` utility. `rcfile` can contain environment commands like set and unset, or a JACL scripts that needs to be run while starting `wadm`. The default file is *~/.wadmrc*
`--no-prompt`	If you specify this option, `wadm` will prompt you for password while executing this command. Use this option if you have defined all passwords in a password file and specified the file using the `--password-file` option.

When you execute the `wadm` command, two things happen:

- A web application with the given *uri_path* and *directory* gets added to the `server.xml` file.
- The WAR file is extracted in the target *directory*.

Example usage of the command is as follows:

```
wadm add-webapp --user=admin --password-file=admin.pwd --host=serverhost
--port=8989 --config=config1 --vs=config1_vs_1 --uri=/testapp /abc/sample.war
```

After you have deployed an application, you can access it from a browser as follows:

`http://`*vs_urlhost*`[:`*vs_port*`]/`*uri_path*`/[`*index_page*`]`

The following table describes the parts of the URL. (The left column lists the part, and the right column describes what the part means.)

TABLE 9–2 Parts of the URL

Part	Description
vs_urlhost	One of the `urlhosts` values for the virtual server.
vs_port	(Optional) Only needed if the virtual server uses a non default port.
uri_path	The same path you used to deploy the application. This is also the context path.
index_page	(Optional) The page in the application that end users are meant to access first.

For example:

```
http://sun.com:80/hello/index.jsp
```

- or -

```
http://sun.com/hello/
```

Deploying Using JSR 88

JSR 88 defines the contracts that enables the tool of multiple providers to configure and deploy applications on any platform product. The implementation requires both tools and Java EE platform products.

You can write you own JSR 88 client to deploy application to the Sun Java System Web Server 7.0. For more information about JSR 88, see `http://jcp.org/en/jsr/detail?id=88`

Managing Web Applications

Once deployed, the application or module exists in the central repository and can be referenced by a number of server instances. Initially, the server instances or clusters you deployed to as targets reference the application or module. To change which server instances and clusters reference an application or module after it is deployed, change an application or module's targets using the Admin Console or change the application references using the `wadm`. Because the application itself is stored in the central repository, adding or deleting targets adds or deletes the same version of an application on different targets. However, an application deployed to more than one target can be enabled on one and disabled on another, so even if an application is referenced by a target, it is not available to you unless it is enabled on that target.Sun Java System Web Server 7.0 allows you to enable or disable a web application. You can do so in either of the following ways:

- "Removing Using the Admin Console Interface" on page 145
- "Removing Using the Command-Line Interface" on page 145
- "Enabling Web Applications" on page 144

Enabling Web Applications

Sun Java System Web Server 7.0 allows you to enable or disable a web application. You can do so in either of the following ways, as discussed in this section:

- "Enabling and Disabling Using the Admin Console" on page 144

Enabling and Disabling Using the Admin Console

To enable or disable a deployed web application using the Admin Console, perform the following steps:

▼ Steps to Enable or Disable a Deployed Web Application

1. **Access the Admin Console, select the server instance, and click the Edit Virtual Server tab.**

2. **Click the Web Applications tab.**

3. **In the Web Applications table, select the web applications you want to enable and click Enable. To disable the application, click Disable.**

4. **Click Save.**

Enabling and Disabling Using the Command-Line Interface

`enable-webapp` command will allow you to enable the web application. `disable-webapp` will allow you to disable the web application. For more information, see `enable-webapp`(1) and `disable-webapp`(1)

Removing Using the Admin Console Interface

To remove a deployed web application using the user interface, perform the following steps:

1. Access the Admin Console, select the server instance, and click the Edit Virtual Server tab.
2. Click the Web Applications tab.
3. In the Web Applications table, select the web applications you want to remove and click the Remove button.
4. When the dialog box appears, click OK.

Removing Using the Command-Line Interface

To remove a deployed web application using the command-line interface, use the following command:

- The `remove web-app` command allows you to remove the web application. For more information on `remove web-app`, see `remove-webapp`(1)

Dynamic Reloading of Web Applications

To set dynamic reloading of web application, you must do the following:

▼ To Set Dynamic Reloading of Web Application

1 **Access the Admin Console. select the server instances, and click the Edit Configuration tab.**

2 **Click the Java tab.**

3 **Click the Servlet Container tab.**

4 **In the Dynamic Reload Interval field. Enter an *integer*, which specifies the interval (in seconds) after which a deployed application will be checked for modifications and reloaded if necessary. To enable dynamic reloading, you must specify a value greater than 0.**

To disable dynamic reloading set the field to `–1`

5 **Click Save.**

To configure through CLI, see `set-servlet-container-prop`(1)

▼ To load a new servlet or reload a deployment descriptor

1 **Create an empty file named `.reload` at the root of the deployed module. For example:**

```
instance_dir/webapps/vs_id/uri/.reload
```

where *vs_id* is the virtual server ID in which the web application is deployed, and *uri* is the value of the `uri` attribute of the `webapp/` element.

2 **Explicitly update the `.reload` file's timestamp (`touch .reload` in UNIX) each time you make the above changes.**

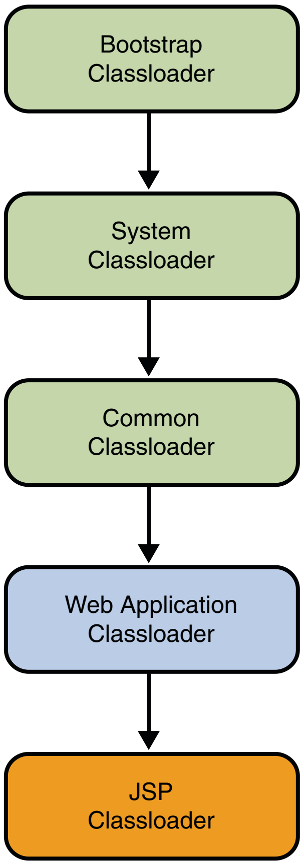
For JSPs, changes are reloaded automatically at a frequency set in the `reload-interval` property of the `jsp-config` element in the `sun-web.xml` file. To disable dynamic reloading of JSPs, set the `reload-interval` property to `-1`.

Classloaders

Understanding Sun Java System Web Server 7.0 classloaders can help you determine where and how you can position supporting JAR and resource files of your modules and applications.

In a Java Virtual Machine (JVM), the classloaders dynamically load a specific Java class file needed for resolving a dependency. For example, when an instance of `java.util.Enumeration` needs to be created, one of the classloaders loads the relevant class into the environment.

Classloaders in the Sun Java System Web Server 7.0 runtime follow the hierarchy shown in the following figure.



🟦	There is a seperate instance of this classloader for each web application
🟧	There is a seperate instance of this classloader per JSP

FIGURE 9–1 Classloader Runtime Hierarchy

This is not a Java inheritance hierarchy, but a delegation hierarchy. In the delegation design, a classloader delegates classloading to its parent before attempting to load a class itself. If the parent classloader cannot load a class, the `findClass()` method is called on the classloader subclass. In effect, a classloader is responsible for loading only the classes not available to the parent.

The exception is the web application classloader, which follows the delegation model in the Servlet specification. The web application Classloader looks in the local classloader before delegating to its parent. You can make the web application Classloader delegate to its parent first by setting `delegate="true"` in the `class-loader` element of the `sun-web.xml` file. For more information, see "Classloader Elements" on page 193.

The following table describes Sun Java System Web Server 7.0 classloaders. The left column lists the classloaders, and the right column lists descriptions of those classloaders and the files they examine.

TABLE 9–3 Sun Java System Web Server 7.0 Classloaders

Classloader	Description
Bootstrap	The Bootstrap Classloader loads the JDK classes. Only one instance of this classloader exists in the entire server.
System	The System Classloader loads the core Sun Java System Web Server 7.0 classes. It is created based on the `class-path-prefix`, `server-class-path`, and `class-path-suffix` attributes of the `<jvm/>` element in the `server.xml` file. The environment classpath is included if `env-classpath-ignored="false"` is set in the `<jvm/>` element. Only one instance of this classloader exists in the entire server. If any changes are made to these attributes or classes, the server must be restarted for the changes to take effect. For more information about the `<jvm/>` element in `server.xml`, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.
Common	The Common Classloader loads classes in the *instance_dir*`/lib/classes` directory, followed by JAR and ZIP files in the *instance_dir*`/lib` directory. The existence of these directories is optional; if they don't exist, the Common Classloader is not created. If any changes are made to these classes, restart the server.
Web Application	The Web Application Classloader loads the servlets and other classes in a specific web application. That is, from `WEB-INF/lib` and `WEB-INF/classes` and from any additional classpaths specified in the `extra-class-path` attribute of the `class-loader` element in `sun-web.xml`. For more information, see "Classloader Elements" on page 193. An instance of this classloader is created for each web application. If dynamic reloading has been enabled, any changes made to these attributes or classes are reloaded by the server without the need for a restart. For more information, see "Dynamic Reloading of Web Applications" on page 145
JSP	The JSP Classloader loads the compiled JSP classes of JSPs. An instance of this classloader is created for each JSP file. Any changes made to a JSP are automatically detected and reloaded by the server, unless dynamic reloading of JSPs has been disabled by setting the `reload-interval` property to `-1` in the `jsp-config` element of the `sun-web.xml` file. For more information, see "jsp-config" on page 194.

CHAPTER 10

Debugging Web Applications

This chapter provides guidelines for debugging web applications in Sun Java System Web Server 7.0, and includes the following sections:

- "Enabling Debugging" on page 151
- "JPDA Options" on page 152
- "Using Developer tools for Debugging" on page 153
- "Debugging JSPs" on page 153
- "Generating a Stack Trace for Debugging" on page 153
- "Using Logging for Debugging" on page 154
- "Using Profiling for Debugging" on page 154

Debugging applications requires you to edit the `server.xml` file as described in this chapter. For more general information, see *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

Enabling Debugging

When you enable debugging, you enable both local and remote debugging. You can enable debugging in one of these ways, as described in this section:

- "Enabling Using the Admin Console" on page 151
- "Enabling by Editing server.xml" on page 152

Sun Java System Web Server debugging is based on the JPDA (Java^{TM} Platform Debugger Architecture software). For more information, see "JPDA Options" on page 152

Enabling Using the Admin Console

Perform the following tasks to enable debugging:

▼ To Enable Debugging

1. **Click the Edit Java Settings tab in the home page.**
2. **Click the JVM Settings tab.**
3. **Select the Enable Debug check box to enable debugging.**

   For more information about debug options, see "JPDA Options" on page 152
4. **Click Save.**

Enabling by Editing server.xml

To enable debugging, set the following attributes of the `jvm` element in the `server.xml` file:

- Set `debug="true"` to turn on debugging.
- Add any desired JPDA debugging options in the `debugoptions` attribute. See "JPDA Options" on page 152.
- To specify the port to use when attaching the JVM to a debugger, specify `address=`*port_number* in the `debugoptions` attribute.

For details about the `server.xml` file, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

JPDA Options

The default JPDA options are as follows:

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n
```

If you change the value of `suspend=y`, the JVM starts in suspended mode and stays suspended until a debugger attaches to it. Use this option if you want to start debugging as soon as the JVM starts.

To specify the port to use when attaching the JVM to a debugger, specify `address=`*port_number*. You can also use shared memory transport `dt_shmem`. It is available only in Win32 platform.

For more information on list of JPDA debugging options, see

`http://java.sun.com/products/jpda/doc/conninv.html#Invocation`

Using Developer tools for Debugging

Sun Java Studio Enterprise 8.1 technology can be used for "remote debugging" if you want to manually attach the IDE to a remote Web Server started in debug mode. The steps are as follows:

▼ Debugging using NetBeans 5.0

- **You can debug web applications by right-clicking the Web project in the IDE and choosing Debug Project from the context menu.**

  Web Server will then start in debug mode and the IDE will pause executing the program at the breakpoint you have set in your application.

Debugging JSPs

When you use Sun Java Studio Enterprise 8.1 to debug JSPs, you can set breakpoints in either the JSP code or the generated servlet code, and you can switch between them and see the same breakpoints in both.

To set up debugging in Sun Java Studio Enterprise 8.1, see "Using Developer tools for Debugging" on page 153.

Generating a Stack Trace for Debugging

You can generate a Java stack trace for debugging as described here:

`http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/`

If the `-Xrs` flag is set (for reduced signal usage) in the `server.xml` file (under `jvm`), comment it out before generating the stack trace. If the `-Xrs` flag is used, the server may simply dump core and restart when you send the signal to generate the trace.

The stack trace goes to the system log file or to stderr based on the `log` attributes in `server.xml`.

For more information about the `server.xml` file, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

Using Logging for Debugging

You can use the Sun Java System Web Server 7.0 log files to help debug your applications. For general information about logging, see the *Sun Java System Web Server 7.0 Administrator's Guide*. For information about `server.xml` file, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

You can change logging settings in one of these ways:

- "Using the Admin Console" on page 154

Using the Admin Console

Perform the following tasks to change the log settings:

▼ To Change the Log settings

1. **Click the View Server Logs tab in the home page.**

2. **Set log preferences as desired.**

3. **Click Save to apply your changes.**

Using Profiling for Debugging

You can use a profiler to perform remote profiling on the Sun Java System Web Server 7.0 to discover choke point in server-side performance. This section describes how to configure these profilers for use with Sun Java System Web Server 7.0:

- "Using the HPROF Profiler" on page 154
- "Using the Optimizeit Profiler" on page 157

Using the HPROF Profiler

HPROF is a simple profiler agent shipped with the Java^{TM} 2 SDK. It is a dynamically linked library that interacts with the Java Virtual Machine Profiler Interface (JVMPI) and writes out profiling information either to a file or to a socket in ASCII or binary format. This information can be further processed by a profiler front-end tool such as HAT.

HPROF can present CPU usage, heap allocation statistics, and monitor contention profiles. In addition, it can also report complete heap dumps and states of all of the monitors and threads in the Java virtual machine. For more details on the HPROF profiler, see the JDK documentation at:

http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html

Once HPROF is installed using the following instructions, its libraries are loaded into the server process.

You can configure Sun Java System Web Server 7.0

▼ HPROF on UNIX

1. **Click Edit Java Settings tab in the home page.**

   JVM General Settings screen appears.

2. **Click Profilers.**

3. **Click New.**

   The Create JVM Profiler popup appears.

4. **Enter the Name and select the JVM Profiler check box to enable the profiler. Leave the Class Path and Native Library Path fields blanks.**

5. **Click New to configure JVM options.**

6. **Click OK.**

   Edit the `server.xml` file as appropriate

```
<! --hprof options -->
<profiler name="hprof" enabled="true"
<jvm-options>
-Xrunhprof:file=log.txt,options
</jvm-options>
</profiler>
```

Note – Do not use the `-Xrs` flag.

Here is an example of options you can use:

```
-Xrunhprof:file=log.txt,thread=y,depth=3
```

The `file` option is important because it determines where the stack dump is written in step 6.

The syntax of HPROF options is as follows:

```
-Xrunhprof[:help]|[:option=value,option2=value2, ...]
```

Using `help` lists options that can be passed to HPROF. The output is as follows:

```
Hprof usage: -Xrunhprof[:help]|[:<option>=<value>, ...]
Option Name and Value Description Default
heap=dump|sites|all heap profiling allcpu=samples|old CPU
usage offformat=a|b ascii or binary output
afile=<file> write data to file java.hprof(.txt for ascii)
net=<host>:<port> send data over a socket write to filedepth=<size>
stack trace depth 4cutoff=<value>
output cutoff point 0.0001lineno=y|n line number in traces?
ythread=y|n thread in traces? ndoe=y|n dump on exit? y
```

7 **Change a line in the Sun Java System Web Server 7.0 start script. The start script file is *install_dir/instance_dir*/`bin/startserv`. Change the following line:**

```
PRODUCT_BIN=webservd-wdog
```

to this:

```
PRODUCT_BIN=webservd
```

8 **Start the server by running the startserv script. As the server runs in the foreground, the command prompt returns only after the server has been stopped.**

9 **Find the process ID of the server process in another window or terminal.**

```
% ps -ef | grep webservd
```

This command lists two `webservd` processes. Look at the PPID (parent process ID) column and identify which of the two processes is the parent process and which is the child process. Note the PID (process ID) of the child process ID.

10 **Send a SIGQUIT signal (signal 3) to the child process:**

```
% kill -QUIT child_PID
```

11 **Run the stopserv script from another window to stop the Web Server.**

```
% ./stopserv
```

This writes an HPROF stack dump to the file you specified using the file HPROF option. For general information about using a stack dump, see "Generating a Stack Trace for Debugging" on page 153.

12 **To return your Web Server to its original configuration, undo the changes in steps 1 and 2.**

Using the Optimizeit Profiler

Information about Optimizeit™ is available at:

```
http://www.borland.com/optimizeit/
```

Once Optimizeit is installed using the following instructions, its libraries are loaded into the server process.

To enable remote profiling with Optimizeit, do one of the following:

- Go to the Common Tasks page in the Admin Console, click the Edit Java Settings tab, click the Profiler link, and edit the following fields before selecting OK:
  - Profiler: Enable
  - Classpath: *Optimizeit_dir*`/lib/optit.jar`
  - Native Lib Path: *Optimizeit_dir*`/lib`
  - JVM Option: For each of these options, type the option in the JVM Option field, select Add, then check its box in the JVM Options list:
    - -DOPTITHOME=*Optimizeit_dir*
    - `-Xrunoii`
    - `-Xbootclasspath/a:`*Optimizeit_dir*`/lib/oibcp.jar`

  Edit the `server.xml` file as appropriate:

```
<!-- Optimizeit options -->
                    <profiler classpath="Optimizeit_dir/lib/optit.jar"
                nativelibrarypath="Optimizeit_dir/lib" enabled="true">
                <jvm-options>
                        -DOPTIT_HOME=Optimizeit_dir -Xboundthreads -Xrunoii
                        -Xbootclasspath/a:Optimizeit_dir/lib/oibcp.jar
             </jvm-options>
        </profiler>
```

  In addition, you may need to set the following in your `server.policy` file:

```
grant codeBase "file:Optimizeit_dir/lib/optit.jar" {
permission java.security.AllPermission;
};
```

  For more information about the `server.policy` file, see "The server.policy File" on page 133.

  When the server starts up with this configuration, you can attach the profiler.

Note – If any of the configuration options are missing or incorrect, the profiler may experience problems that affect the performance of the Sun Java System Web Server 7.0.

◆ ◆ ◆ APPENDIX A

Deployment Descriptor Files

This chapter includes the following sections:

ser

About Deployment Descriptor Files

The deployment descriptor conveys the elements and configuration information of a web application between application developers, application assemblers, and deployers. For Java Servlets v.2.4, the deployment descriptor is defined in terms of an XML schema document.

Migration Issues

Migration will create a detailed log in the user-specified log file. For every instance it will create a log in the following syntax, `MIGRATION_<server instance name>_MMM_DD_YYYY_HH_MM_AM/PM.log`.

If there is no log directory, then it will store the log file at `<install_dir>/admin-server/logs`.

Java EE Standard Descriptors

sun-web.xml

You define roles in the deployment descriptor file `web.xml` and the corresponding role mappings in the `sun-web.xml` deployment descriptor file for individually deployed web modules.

default-web.xml

`default-web.xml` is a global web deployment descriptor file that is shared by deployed web applications. It is used to configure the `DefaultServlet` and `JspServlet`. In addition, it specifies: MIME mappings based on extensions Welcome files Global filters and security constraints Individual web applications inherit and might override the configuration settings inherited from `default-web.xml` with their own `web.xml`. There is one `default-web.xml`per server instance. It is shared by all web applications deployed on the server instance. Depending on the configuration capabilities of the hosting application, a virtual server may replace the server-wide `default-web.xml` with its own. In that case, a virtual server's `default-web.xml` is shared by all webapps deployed on the virtual server.

Sun Java System Web Server Descriptors

The sun-web-app_2_4-1.dtd File

The `sun-web-app_2_4-1.dtd` file defines the structure of the `sun-web.xml` file, including the elements it can contain and the subelements and attributes these elements can have. The `sun-web-app_2_4-1.dtd` file is located in the *install_dir*`/lib/dtds` directory.

Note – Do not edit the `sun-web-app_2_4-1.dtd` file. Its contents change only with new versions of Sun Java System Web Server 7.0.

For general information about DTD files and XML, see the XML specification at:

`http://www.w3.org/TR/2004/REC-xml-20040204`

Each element defined in a DTD file (which may be present in the corresponding XML file) can contain the following:

- "Subelements" on page 161

Subelements

Elements can contain subelements. For example, the following file fragment defines the cache element.

```
<!ELEMENT cache (cache-helper*, default-helper?, property*, cache-mapping*)>
```

The `ELEMENT` tag specifies that a `cache` element can contain `cache-helper`, `default-helper`, `property`, and `cache-mapping` subelements.

The following table shows how optional suffix characters of subelements determine the requirement rules, or number of allowed occurrences, for the subelements. The left column lists the subelement ending character, and the right column lists the corresponding requirement rule.

TABLE A–1 Requirement Rules and Subelement Suffixes

Subelement Suffix	Requirement Rule
*element**	Can contain *zero or more* of this subelement.
element?	Can contain *zero or one* of this subelement.
element+	Must contain *one or more* of this subelement.
element (no suffix)	Must contain *only one* of this subelement.

If an element cannot contain other elements, you see `EMPTY` or `(#PCDATA)` instead of a list of element names in parentheses.

Data

Some elements contain character data instead of subelements. These elements have definitions of the following format:

```
<!ELEMENT element-name (#PCDATA)/>
```

For example:

```
<!ELEMENT description (#PCDATA)/>
```

In the `sun-web.xml` file, white space is treated as part of the data in a data element. Therefore, there should be no extra white space before or after the data delimited by a data element. For example:

```
<description/>class name of session manager</description>
```

Attributes

Elements that have ATTLIST tags contain attributes (name-value pairs). For example:

```
<!ATTLIST     cachemax-capacity CDATA     "4096"
              timeout           CDATA     "30"
              enabled           %boolean; "false">
A cache element can contain max-capacity, timeout, and enabled attributes.
```

The #REQUIRED label means that a value must be supplied. The #IMPLIED label means that the attribute is optional, and that Sun Java System Web Server 7.0 generates a default value. Wherever possible, explicit defaults for optional attributes (such as "true") are listed.

Attribute declarations specify the type of the attribute. For example, CDATA means character data, and %boolean is a predefined enumeration.

Elements in the sun-web.xml File

This section describes the XML elements in the sun-web.xml file. Elements are grouped as follows:

- "General Elements" on page 163
- "Security Elements" on page 167
- "Session Elements" on page 170
- "Reference Elements" on page 176
- "Caching Elements" on page 183
- "Classloader Elements" on page 193
- "JSP Elements" on page 194
- "Internationalization Elements" on page 197

This section also includes an alphabetical list of the elements for quick reference. See "Alphabetical List of sun-web.xml Elements" on page 201.

Note – Subelements must be defined in the order in which they are listed under each Subelements heading, unless otherwise noted.

Note – Each `sun-web.xml` file must begin with the following `DOCTYPE` header:

```
<!DOCTYPE sun-web-app PUBLIC -//Sun Microsystems, Inc.//DTD Application Server
8.1 Servlet 2.4//EN"
http://www.sun.com/software/appserver/dtds/sun-web-app_2_4-1.dtd">
```

For an alphabetical list of elements in `sun-web.xml`, see "Alphabetical List of sun-web.xml Elements" on page 201

General Elements

General elements are as follows:

- "sun-web-app" on page 163
- "property" on page 166
- "description" on page 167

sun-web-app

Defines Sun Java System Web Server 7.0- specific configuration for a web application. This is the root element; there can only be one `sun-web-app` element in a `sun-web.xml` file.

Subelements

The following table describes subelements for the `sun-web-app` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–2 `sun-web-app` Subelements

Element	Required	Description
"context-root" on page 166	zero or more	Contains the web context roots for the web application.
"servlet" on page 168	zero or more	Specifies a principal name for a servlet, which is used for the run-as role defined in `web.xml`.
"session-config" on page 170	zero or one	Specifies session manager, session cookie, and other session-related information.
"resource-env-ref" on page 176	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding Java EE XML file.

TABLE A–2 `sun-web-app` Subelements *(Continued)*

Element	Required	Description
"resource-ref" on page 181	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding Java EE XML file.
"service-ref" on page 177	zero or more	Specifies runtime settings for a web service reference
"cache" on page 184	zero or one	Configures caching for web application components.
"class-loader" on page 193	zero or one	Specifies classloader configuration information.
"jsp-config" on page 194	zero or one	Specifies JSP configuration information.
"locale-charset-info" on page 198	zero or one	Specifies internationalization settings.
"property" on page 166	zero or more	Specifies a property, which has a name and a value.
"message-destination-name" on page 200	zero or more	Specifies a logical message destination.
"webservice-description" on page 200	zero or more	Specifies a web service description

Attributes

none

Properties

The following table describes properties for the `sun-web-app` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–3 `sun-web-app` Properties

Property Name	Default Value	Description
`crossContextAllowed`	`true`	If `true`, allows this web application to access the contexts of other web applications using the `ServletContext.getContext()` method.

TABLE A–3 `sun-web-app` Properties *(Continued)*

Property Name	Default Value	Description
`encodeCookies`	`true`	If `true`, Sun Java System Web Server 7.0 URL encodes cookies before sending them to the client. If you don't want cookies to be encoded, add the following to `sun-web.xml`: `<property name="encodeCookies" value="false"/>` For the above example, put the line directly under the `<sun-web-app>` tag; do not embed this in any other tag.
`tempdir`	*instance_dir*/`generated/`	Specifies a temporary directory for use by this web application. This value is used to construct the value of the `javax.servlet.context.tempdir` context attribute. Compiled JSPs are also placed in this directory.
`singleThreadedServletPoolSize`	`5`	Specifies the maximum number of servlet instances allocated for each `SingleThreadModel` servlet in the web application.
`reuseSessionID`	`false`	If `true`, this property causes the web application to reuse the `JSESSIONID` value (if present) in the request header as the session ID when creating sessions. The default behavior of web applications is to not reuse session IDs and instead generate cryptographically random session IDs for new sessions.

TABLE A–3 `sun-web-app` Properties *(Continued)*

Property Name	Default Value	Description
`relativeRedirectAllowed`	`false`	If `true`, allows the web application to send a relative URL to the client using the `HttpServletResponse.sendRedirect()` API (that is, it suppresses the container from translating a relative URL to a fully qualified URL).

context-root

Contains the web context root of the application or web applications. Overrides the corresponding element in the `web.xml` file.

Subelements

none

Attributes

none

property

Specifies a property, which has a name and a value. A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Sun Java System Web Server 7.0.
- Needed by a system or object that Sun Java System Web Server 7.0 doesn't have knowledge of, such as an LDAP server or a Java class.

For example, a `manager-properties` element can include property subelements:

```
<manager-properties>
        <property name="reapIntervalSeconds" value="20" />
     </manager-properties>
```

Which properties a `manager-properties` element uses depends on the value of the parent `session-manager` elements `persistence-type` attribute. For details, see the description of the `session-manager` element.

Subelements

The following table describes subelements for the `property` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–4 `property` Subelements

Element	Required	Description
"description" on page 167	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the `property` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–5 `property` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the property or variable.
`value`	none	Specifies the value of the property or variable.

description

Contains a text description of the parent element.

Subelements

none

Attributes

none

Security Elements

Security elements are as follows:

- "security-role-mapping" on page 167
- "servlet" on page 168
- "servlet-name" on page 169
- "role-name" on page 169
- "principal-name" on page 169
- "group-name" on page 169

security-role-mapping

Maps roles to users or groups in the currently active realm.

Subelements

The following table describes subelements for the `security-role-mapping` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–6 `security-role-mapping` Subelements

Element	Required	Description
"role-name" on page 169	only one	Contains the role name.
"principal-name" on page 169	requires at least one `principal-name` or `group-name`	Contains a principal (user) name in the current realm.
"group-name" on page 169	requires at least one `principal-name` or `group-name`	Contains a group name in the current realm.

Attributes

none

servlet

Specifies a principal name for a servlet, which is used for the `run-as` role defined in `web.xml`.

Subelements

The following table describes subelements for the `servlet` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–7 `servlet` Subelements

Element	Required	Description
"servlet-name" on page 169	only one	Contains the name of a servlet, which is matched to a `servlet-name` in `web.xml`.
"principal-name" on page 169	only one	Contains a principal (user) name in the current realm.

Attributes

none

servlet-name

Contains data that specifies the name of a servlet, which is matched to a `servlet-name` in `web.xml`. This name must be present in `web.xml`.

Subelements

none

Attributes

none

role-name

Contains data that specifies the `role-name` in the `security-role` element of the `web.xml` file.

Subelements

none

Attributes

none

principal-name

Contains data that specifies a principal (user) name in the current realm.

Subelements

none

Attributes

none

group-name

Contains data that specifies a group name in the current realm.

Subelements

none

Attributes

none

Session Elements

Session elements are as follows:

- "session-config" on page 170
- "session-manager" on page 170
- "manager-properties" on page 171
- "store-properties" on page 173
- "session-properties" on page 174
- "cookie-properties" on page 175

Note – The session manager interface is unstable. An unstable interface may be experimental or transitional, and thus may change incompatibly, be removed, or be replaced by a more stable interface in the next release.

---

session-config

Specifies session configuration information.

Subelements

The following table describes subelements for the `session-config` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–8 `session-config` Subelements

Element	Required	Description
"session-manager" on page 170	zero or one	Specifies session manager configuration information.
"session-properties" on page 174	zero or one	Specifies session properties.
"cookie-properties" on page 175	zero or one	Specifies session cookie properties.

Attributes

none

session-manager

Specifies session manager information.

Note – As of Sun Java System Web Server 7.0, you cannot define a session manager either for a single sign-on session or for a virtual server. You must define session managers at the level of web applications.

Subelements

The following table describes subelements for the `session-manager` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–9 `session-manager` Subelements

Element	Required	Description
"manager-properties" on page 171	zero or one	Specifies session manager properties.
"store-properties" on page 173	zero or one	Specifies session persistence (storage) properties.

Attributes

The following table describes attributes for the `session-manager` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–10 `session-manager` Attributes

Attribute	Default Value	Description
`persistence-type`	`memory`	(Optional) Specifies the session persistence mechanism. Allowed values are `memory`, `file`, `s1ws60`, and `mmap`. Setting the value of persistence type to `memory` is equivalent to using Sun Java System Web Server 7.0's `IWS60` without any store. Setting the value of persistence type to `file` is equivalent to using Sun Java System Web Server 7.0's `IWS60` with `FileStore`.

manager-properties

Specifies session manager properties.

Subelements

The following table describes subelements for the `manager-properties` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–11 `manager-properties` Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the `manager-properties` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–12 `manager-properties` Properties

Property Name	Default Value	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions. Setting this value lower than the frequency at which session data changes is recommended. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed web site, or you could lose the last few hits each time you restart the server.
`maxSessions`	`-1`	Specifies the maximum number of active sessions, or `-1` (the default) for no limit.
`sessionFilename`	none; state is not preserved across restarts	Specifies the absolute or relative path name of the file in which the session state is preserved between application restarts, if preserving the state is possible. A relative path name is relative to the temporary directory for this web application. Applicable only if the `persistence-type` attribute of the "session-manager" on page 170 element is `memory`.

store-properties

Specifies session persistence (storage) properties.

Subelements

The following table describes subelements for the `store-properties` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–13 `store-properties` Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the `store-properties` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–14 `store-properties` Properties

Property Name	Default Value	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions for those sessions that are currently swapped out. Setting this value lower than the frequency at which session data changes is recommended. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed web site, or you could lose the last few hits each time you restart the server.
`directory`	directory specified by `javax.servlet.context` `.tempdir` context attribute	Specifies the absolute or relative path name of the directory into which individual session files are written. A relative path is relative to the temporary work directory for this web application.

session-properties

Specifies session properties.

Subelements

The following table describes subelements for the `session-properties` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–15 `session-properties` Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the `session-properties` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–16 session-properties Properties

Property Name	Default Value	Description
`timeoutSeconds`	`600`	Specifies the default maximum inactive interval (in seconds) for all sessions created in this web application. If set to `0` or less, sessions in this web application never expire. If a session-timeout element is specified in the `web.xml` file, the `session-timeout` value overrides any `timeoutSeconds` value. If neither `session-timeout` nor `timeoutSeconds` is specified, the `timeoutSeconds` default is used. Note that the `session-timeout` element in `web.xml` is specified in minutes, not seconds.
`enableCookies`	`true`	Uses cookies for session tracking if set to `true`.

TABLE A–16 session-properties Properties (Continued)

Property Name	Default Value	Description
enableURLRewriting	true	Enables URL rewriting. This provides session tracking via URL rewriting when the browser does not accept cookies. You must also use an encodeURL or encodeRedirectURL call in the servlet or JSP.

cookie-properties

Specifies session cookie properties.

Subelements

The following table describes subelements for the cookie-properties element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–17 cookie-properties Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the cookie-properties element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–18 cookie-properties Properties

Property Name	Default Value	Description
cookiePath	context path at which the web application is installed.	Specifies the path name that is set when the session tracking cookie is created. The browser sends the cookie if the path name for the request contains this path name. If set to / (slash), the browser sends cookies to all URLs served by the Sun Java System Web Server 7.0. You can set the path to a narrower mapping to limit the request URLs to which the browser sends cookies.

TABLE A–18 `cookie-properties` Properties *(Continued)*

Property Name	Default Value	Description
`cookieMaxAgeSeconds`	`-1`	Specifies the expiration time (in seconds) after which the browser expires the cookie. The default value of `-1` indicates that the cookie never expires.
`cookieDomain`	(unset)	Specifies the domain for which the cookie is valid.
`cookieComment`	Sun Java System Web Server session tracking cookie	Specifies the comment that identifies the session tracking cookie in the cookie file. Applications can provide a more specific comment for the cookie.

Reference Elements

Reference elements are as follows:

- "resource-env-ref" on page 176
- "resource-env-ref-name" on page 177
- "resource-ref" on page 181
- "service-ref" on page 177
- "res-ref-name" on page 182
- "default-resource-principal" on page 182
- "name" on page 183
- "password" on page 183
- "jndi-name" on page 183

resource-env-ref

Maps the "res-ref-name" on page 182 in the corresponding Java EE `web.xml` file `resource-env-ref` entry to the absolute `jndi-name` of a resource.

Subelements

The following table describes subelements for the `resource-env-ref` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–19 `resource-env-ref` Subelements

Element	Required	Description
"resource-env-ref-name" on page 177	only one	Specifies the `res-ref-name` in the corresponding Java EE `web.xml` file `resource-env-ref` entry.
"jndi-name" on page 183	only one	Specifies the absolute `jndi-name` of a resource.

Attributes

none

resource-env-ref-name

Contains data that specifies the "res-ref-name" on page 182 in the corresponding Java EE `web.xml` file `resource-env-ref` entry.

Subelements

none

Attributes

none

service-ref

Specifies the runtime settings for a web service reference. Runtime information is only needed in the following cases:

- To define the port used to resolve a container-managed port
- To define the default Stub/Call property settings for Stub objects
- To define the URL of a final WSDL document to be used instead of the one associated with `service-ref` in the standard Java EE deployment descriptor.

TABLE A–20 `service-ref` Subelements

Element	Required	Description
"service-ref-name" on page 178	only one	Specifies the web service reference name relative to `java:comp/env`.
"port-info" on page 178	zero or more	Specifies information for a port within a web service reference.
"call-property" on page 180	zero or more	Specifies JAX-RPC property values that can be set on a `javax.xml.rpc.Call` object before it is returned to the web service client.
"service-impl-class" on page 181	zero or more	Specifies the name of the generated service implementation class.
"service-qname" on page 181	zero or one	Specifies the WSDL service element that is being referenced.

service-ref-name

Specifies the web service reference name relative to `java:comp/env`.

Subelements

none

Attributes

none

port-info

Either a `service-endpoint-interface` or a `wsdl-port` or both must be specified. If both are specified, `wsdl-port` specifies the port that the container chooses for container-managed port selection. The same `wsdl-port` value must not appear in more than one `port-info` element within the same `service-ref`. If a `service-endpoint-interface` is using container-managed port selection, its value must not appear in more than one `port-info` element within the same `service-ref`.

Subelements

The following elements describes subelements for the `port-info` element

TABLE A–21 `port-info` Subelements

Element	Required	Description
"service-endpoint-interface" on page 179	zero or one	Specifies the web service reference name relative to `java:comp/env`.
"wsdl-port" on page 179	zero or one	Specifies the WSDL port.
"stub-property" on page 180	zero or one	Specifies JAX-RPC property values that are set on `javax.xml.rpc.Stub` object before it is returned tot he web service client.
"call-property" on page 180	zero or one	Specifies JAX-RPC property values that are set on `javax.xml.rpc.Stub` object before it is returned tot he web service client.
"`message-security-binding` Element" on page 125	zero or one	Specifies a custom authentication provider binding.

service-endpoint-interface

Specifies the web service reference name relative to `java:comp/env`.

Subelements

none

Attributes

none

wsdl-port

Specifies the WSDL port.

Subelements

The following table describes subelements for the `wsdl-port` element

TABLE A–22 `wsdl-port` Subelements

Element	Required	Description
"namespaceURI" on page 179	only one	Specifies the namespace URI.
"localpart" on page 179	only one	Specifies the local part of a QNAME.

namespaceURI

Specifies the namespace URI.

Subelements

none

Attributes

none

localpart

Specifies the local part of a QNAME.

Subelements

none

Attributes

none

stub-property

Specifies JAX-RPC property values that are set on a `javax.xml.rpc`. Stub object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC Stub implementation.

Subelements

The following table describes subelements for the `stub-property` element.

TABLE A–23 `stub-property` subelements

Element	Required	Description
"name" on page 183	only one	Specifies the name of the entity.
"value" on page 193	only one	Specifies the value of the entity.

call-property

Specifies JAX-RPC property values that can be set on a `javax.xml.rpc`. Call object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC Call implementation.

TABLE A–24 `call-property` subelements

Element	Required	Description
"name" on page 183	only one	Specifies the name of the entity.
"value" on page 193	only one	Specifies the value of the entity.

wsdl-override

Specifies a valid URL pointing to a final WSDL document. If not specified, the WSDL document associated with the `service-ref` in the standard J2EE deployment descriptor is used.

Subelements

none

Attributes

none

service-impl-class

Specifies the name of the generated service implementation class.

Subelements

none

Attirbutes

none

service-qname

Specifies the WSDL service element that is being referred to

Subelements

The following table describes subelements for the `service-qname` element

TABLE A–25 `service-qname` Subelements

Element	Required	Description
"namespaceURI" on page 179	only one	Specifies the namespace URI.
"localpart" on page 179	only one	Specifies the local part of a QNAME.

resource-ref

Maps the "res-ref-name" on page 182 in the corresponding Java EE `web.xml` file `resource-ref` entry to the absolute "jndi-name" on page 183 of a resource.

Subelements

The following table describes subelements for the `resource-ref` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–26 `resource-ref` Subelements

Element	**Required**	**Description**
"res-ref-name" on page 182	only one	Specifies the `res-ref-name` in the corresponding Java EE `web.xml` file `resource-ref` entry.

TABLE A–26 `resource-ref` Subelements *(Continued)*

Element	Required	Description
"jndi-name" on page 183	only one	Specifies the absolute `jndi-name` of a resource.
"default-resource-principal" on page 182	zero or one	Specifies the default principal (user) for the resource.

Attributes

none

res-ref-name

Contains data that specifies the `res-ref-name` in the corresponding Java EE `web.xml` file `resource-ref` entry.

Subelements

none

Attributes

none

default-resource-principal

Specifies the default principal (user) for the resource.

If this element is used in conjunction with a JMS Connection Factory resource, the name and password subelements must be valid entries in Message Queue's broker user repository.

Subelements

The following table describes subelements for the `default-resource-principal` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–27 `default-resource-principal` Subelements

Element	Required	Description
"name" on page 183	only one	Contains the name of the principal.
"password" on page 183	only one	Contains the password for the principal.

Attributes

none

name

Contains data that specifies the name of the principal.

Subelements

none

Attributes

none

password

Contains data that specifies the password for the principal.

Subelements

none

Attributes

none

jndi-name

Contains data that specifies the absolute `jndi-name` of a URL resource or a resource in the `server.xml` file.

Note – To avoid collisions with names of other enterprise resources in JNDI, and to avoid portability problems, all names in a Sun Java System Web Server 7.0 application should begin with the string `java:comp/env.`

Subelements

none

Attributes

none

Caching Elements

For details about response caching as it pertains to servlets, see "Caching Servlet Results" on page 56 and "JSP Cache Tags" on page 72.

Caching elements are as follows:

- “cache” on page 184
- “cache-helper” on page 186
- “default-helper” on page 187
- “cache-mapping” on page 188
- “url-pattern” on page 189
- “cache-helper-ref” on page 189
- “timeout” on page 190
- “refresh-field” on page 190
- “http-method” on page 191
- “key-field” on page 191
- “constraint-field” on page 192
- “value” on page 193

cache

Configures caching for web application components.

Subelements

The following table describes subelements for the `cache` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–28 `cache` Subelements

Element	Required	Description
“cache-helper” on page 186	zero or more	Specifies a custom class that implements the `CacheHelper` interface.
“default-helper” on page 187	zero or one	Allows you to change the properties of the default, built-in `cache-helper` class.
“property” on page 166	zero or more	Specifies a cache property, which has a name and a value.
“cache-mapping” on page 188	zero or more	Maps a URL pattern or a servlet name to its cacheability constraints.

Attributes

The following table describes attributes for the `cache` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–29 cache Attributes

Attribute	Default Value	Description
max-entries	4096	(Optional) Specifies the maximum number of entries the cache can contain. Must be a positive integer.
timeout-in-seconds	30	(Optional) Specifies the maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. Can be overridden by a timeout element.
enabled	false	(Optional) Determines whether servlet and JSP caching is enabled. Legal values are on, off, yes, no, 1, 0, true, false.

Properties

The following table describes properties for the cache element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–30 cache Properties

Property Name	Default Value	Description
cacheClassName	com.sun.appserv.web .cache.LruCache	Specifies the fully qualified name of the class that implements the cache functionality. For a list of valid values, see“Cache Class Names” on page 186.
MultiLRUSegmentSize	4096	Specifies the number of entries in a segment of the cache table that should have its own LRU (least recently used) list. Applicable only if cacheClassName is set to com.sun.appserv.web .cache.MultiLruCache.
MaxSize	unlimited; Long.MAX_VALUE	Specifies an upper bound on the cache memory size in bytes (KB or MB units). Example values are 32 KB or 2 MB. Applicable only if cacheClassName is set to com.sun.appserv.web. cache.BoundedMultiLruCache.

Cache Class Names

The following table lists possible values of the `cacheClassName` property. The left column lists the value, and the right column describes the kind of cache the value specifies.

TABLE A–31 `cacheClassName` Values

Value	Description
`com.sun.appserv.web.cache.LruCache`	A bounded cache with an LRU (least recently used) cache replacement policy.
`com.sun.appserv.web.cache.BaseCache`	An unbounded cache suitable if the maximum number of entries is known.
`com.sun.appserv.web.cache.MultiLruCache`	A cache suitable for a large number of entries (>4096). Uses the `MultiLRUSegmentSize` property.
`com.sun.appserv.web.cache.BoundedMultiLruCache`	A cache suitable for limiting the cache size by memory rather than number of entries. Uses the `MaxSize` property.

cache-helper

Specifies a class that implements the `CacheHelper` interface. For details, see "CacheHelper Interface" on page 58.

Subelements

The following table describes subelements for the `cache-helper` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–32 `cache-helper` Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `cache-helper` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–33 `cache-helper` Attributes

Attribute	Default Value	Description
`name`	default	Specifies a unique name for the helper class, which is referenced in the cache-mapping element.
`class-name`	none	Specifies the fully qualified class name of the cache helper, which must implement the `com.sun.appserv.web.CacheHelper` interface.

default-helper

Allows you to change the properties of the built-in default `cache-helper` class.

Subelements

The following table describes subelements for the `default-helper` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–34 `default-helper` Subelements

Element	Required	Description
"property" on page 166	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the `default-helper` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–35 `default-helper` Properties

Property Name	Default Value	Description
`cacheKeyGeneratorAttrName`	Uses the built-in default `cache-helper` key generation, which concatenates the servlet path with `key-field` values, if any.	The caching engine looks in the `ServletContext` for an attribute with a name equal to the value specified for this property to determine whether a customized `CacheKeyGenerator` implementation is used. An application may provide a customized key generator rather than using the default helper. See "CacheKeyGenerator Interface" on page 61 .

cache-mapping

Maps a URL pattern or a servlet name to its cacheability constraints.

Subelements

The following table describes subelements for the `cache-mapping` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–36 `cache-mapping` Subelements

Element	Required	Description
"servlet-name" on page 169	requires one `servlet-name` or `url-pattern`	Contains the name of a servlet.
"url-pattern" on page 189	requires one `servlet-name` or `url-pattern`	Contains a servlet URL pattern for which caching is enabled.
"cache-helper-ref" on page 189	required if `timeout`, `refresh-field`,`http-method`, `key-field`, and `constraint-field` are not used	Contains the name of the `cache-helper` used by the parent `cache-mapping` element.
"cache-mapping" on page 188	zero or more	Specifies the RequestDispatcher methods for which caching is to be enabled on the target resource. Valid values are REQUEST, FORWARD, INCLUDE, and ERROR (default: REQUEST).
"timeout" on page 190	zero or one if `cache-helper-ref` is not used	Contains the `cache-mapping` specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed

TABLE A–36 `cache-mapping` Subelements *(Continued)*

Element	Required	Description
"refresh-field" on page 190	zero or one if `cache-helper-ref` is not used	Specifies a field that gives the application component a programmatic way to refresh a cached entry.
"http-method" on page 191	zero or more if `cache-helper-ref` is not used	Contains an HTTP method that is eligible for caching.
"key-field" on page 191	zero or more if `cache-helper-ref` is not used	Specifies a component of the key used to look up and extract cache entries.
"constraint-field" on page 192	zero or more if `cache-helper-ref` is not used	Specifies a cacheability constraint for the given `url-pattern` or `servlet-name`.

Attributes

none

url-pattern

Contains data that specifies a servlet URL pattern for which caching is enabled. See the Java Servlet 2.4 specification, section SRV 11.2 for, applicable patterns.

Subelements

none

Attributes

none

cache-helper-ref

Contains data that specifies the name of the `cache-helper` used by the parent `cache-mapping` element.

Subelements

none

Attributes

none

timeout

Contains data that specifies the `cache-mapping` specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. If not specified, the default is the value of the `timeout` attribute of the `cache` element.

Subelements

none

Attributes

The following table describes attributes for the `timeout` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–37 `timeout` Attributes

Attribute	Default Value	Description
`name`	none	Specifies the timeout input parameter, whose value is interpreted in seconds. The field's type must be `java.lang.Long` or `java.lang.Integer`.
`scope`	`context.attribute`	(Optional) Specifies the scope in which the input parameter can be present. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

refresh-field

Specifies a field that gives the application component a programmatic way to refresh a cached entry.

Subelements

none

Attributes

The following table describes attributes for the `refresh-field` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–38 `refresh-field` Attributes

Attribute	Default Value	Description
`name`	none	Specifies the input parameter name. If the parameter is present in the specified scope and its value is `true`, the cache will be refreshed.
`scope`	`request.parameter`	(Optional) Specifies the scope in which the input parameter can be present. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

http-method

Contains data that specifies an HTTP method that is eligible for caching. The default is `GET`.

Subelements

none

Attributes

none

key-field

Specifies a component of the key used to look up and extract cache entries. The web container looks for the named parameter, or field, in the specified scope.

If this element is not present, the web container uses the Servlet Path (the path section that corresponds to the servlet mapping that activated the current request). See the Servlet 2.4 specification, section SRV 4.4, for details on the Servlet Path.

Subelements

none

Attributes

The following table describes attributes for the `key-field` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–39 key-field Attributes

Attribute	Default Value	Description
name	none	Specifies the input parameter name.
scope	request.parameter	(Optional) Specifies the scope in which the input parameter can be present. Allowed values are context.attribute, request.header, request.parameter, request.cookie, session.id, and session.attribute.

constraint-field

Specifies a cache ability constraint for the given url-pattern or servlet-name.

All constraint-field constraints must pass for a response to be cached. If there are value constraints, at least one of them must pass.

Subelements

The following table describes subelements for the constraint-field element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–40 constraint-field Subelements

Element	Required	Description
"value" on page 193	zero or more	Contains a value to be matched to the input parameter value.

Attributes

The following table describes attributes for the constraint-field element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–41 constraint-field Attributes

Attribute	Default Value	Description
name	none	Specifies the input parameter name.

TABLE A–41 `constraint-field` Attributes *(Continued)*

Attribute	Default Value	Description
`scope`	`request.parameter`	(Optional) Specifies the scope in which the input parameter can be present. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.
`cache-on-match`	true	(Optional) If `true`, caches the response if matching succeeds. Overrides the same attribute in a `value` subelement.
`cache-on-match-failure`	false	(Optional) If `true`, caches the response if matching fails. Overrides the same attribute in a `value` subelement.

value

Specifies the value of the entity

Subelements

none

Attributes

none

Classloader Elements

Classloader elements are as follows:

- "class-loader" on page 193

class-loader

Configures the classloader for the web application.

Subelements

none

Attributes

The following table describes attributes for the `class-loader` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–42 `class-loader` Attributes

Attribute	Default Value	Description
`extra-class-path`	null	(Optional) Specifies additional classpath settings for this web application.
`delegate`	`false`	(Optional) If `true`, the web application follows the standard classloader delegation model and delegates to its parent classloader first before looking in the local classloader. If `false`, the web application follows the delegation model specified in the Servlet specification and looks in its classloader before looking in the parent classloader. For a web component of a web service, you must set this value to `true`. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`dynamic-reload-interval`	value of the `dynamicreloadinterval` attribute of the `<jvm>` element in `server.xml`	(Optional) Allows an application to override the `dynamicreloadinterval` setting in `server.xml`. Specifies the frequency (in seconds) at which a web application is checked for modifications, and then reloaded if modifications have been made. Setting this value to less than or equal to 0 disables dynamic reloading of the application. If not specified, the value from `server.xml` is used. For more information about `server.xml`, see *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

JSP Elements

JSP elements are as follows:

- “jsp-config” on page 194

jsp-config

Specifies JSP configuration information allows web application to customize the compilation and execution of its JSP's.

Subelements

The following table describes subelements for the `jsp-config` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–43 `jsp-config` Subelements

Element	Required	Description
"name" on page 183	zero or more	Specifies a property.

Attributes

none

Properties

The following table describes properties for the `jsp-config` element. The left column lists the property name, the middle column indicates the default value, and the right column describes what the property does.

TABLE A–44 `jsp-config` Properties

Property Name	Default Value	Description
`ieClassId`	`clsid:8AD9C840-044E-11D1-B3E9-00805F499D93`	The Java plug-in COM class ID for Internet Explorer. Used by the <jsp:plugin> tags.

TABLE A–44 jsp-config Properties *(Continued)*

Property Name	Default Value	Description
javaCompilerPlugin	internal JDK compiler (javac)	This property is deprecated in this release. By this, we mean this is supported in 7.0 but will NOT be supported in future release If JSP Pages import classes from unnamed packages, the default-JDK compiler will throw a compile time error when JSP- generated servlets are compiled. To compile JSP- generated servlets, set the javaCompilerPlugin property to org.apache.jasper.compiler.SunJavaCompiler. Note: The jspc command-line compiler for JSPs no longer supports -javac option. Since this property is deprecated, you are strongly encouraged to modify JSPs so that the imported classes have a package name. See also the -deprecatedjavac switch of jspc, described in "Compiling JSPs Using the Command-Line Compiler" on page 69.
javaEncoding	UTF8	Specifies the encoding for the generated Java servlet. This encoding is passed to the Java compiler used to compile the servlet as well. By default, the web container tries to use UTF8. If that fails, it tries to use the javaEncoding value. For encodings you can use, see: http://java.sun.com/ j2se/1.5.0/docs/guide/intl/encoding.doc.html
classdebuginfo	false	Specifies whether the generated Java servlets should be compiled with the debug option set (-g for javac).
keepgenerated	true	If set to true, keeps the generated Java files. If false, deletes the Java files.
mappedfile	false	If set to true, generates separate write calls for each HTML line and comments that describe the location of each line in the JSP file. By default, all adjacent write calls are combined and no location comments are generated.
scratchdir	Default work directory for the web application	The working directory created for storing all of the generated code.

TABLE A–44 `jsp-config` Properties *(Continued)*

Property Name	Default Value	Description
`reload-interval`	`0`	Specifies the frequency (in seconds) at which JSP files are checked for modifications. Setting this value to 0 checks JSPs for modifications on every request. Setting this value to -1 disables checks for JSP modifications and JSP recompilation.
`initial-capacity`	`32`	Specifies the initial size of the hash table of compiled JSP classes (see the following example).

The following example illustrates the use of the `initial-capacity` property described in the table above. The example shows how you would configure a value of 1024:

```
<jsp-config> <property name=”initial-capacity” value=”1024” /></jsp-config>
```

Internationalization Elements

Internationalization elements are as follows:

- “parameter-encoding” on page 197
- “locale-charset-info” on page 198
- “locale-charset-map” on page 199

parameter-encoding

Specifies a hidden field or default charset that determines the character encoding the web container uses to decode parameters for `request.getParameter` calls when the charset is not set in the request's `Content-Type`.

For encodings you can use, see:

`http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html`

Attributes

The following table describes attributes for the `parameter-encoding` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–45 parameter-encoding Attributes

Attribute	Default Value	Description
`form-hint-field`	none	The value of the hidden field in the form that specifies the parameter encoding.
`default-charset`	none	This value is used for parameter encoding if neither `request.setCharacterEncoding()` is called nor `form-hint-field` is found in the request.

Subelements

none

Attributes

none

locale-charset-info

Specifies the mapping between the locale and the character encoding that should be set in the `Content-type` header of the response if a servlet or JSP sets the response locale using the `ServletResponse.setLocale` method. This overrides the web container's default `locale-to-charset` mapping.

Subelements

The following table describes subelements for the `locale-charset-info` element. The left column lists the subelement name, the middle column indicates the requirement rule, and the right column describes what the element does.

TABLE A–46 locale-charset-info Subelements

Element	Required	Description
"locale-charset-map" on page 199	one or more	Maps a locale to a character set.
"parameter-encoding" on page 197	zero or one	Deprecated. Use the `parameter-encoding` element under `sun-web-app` instead. This is supported only for backward compatibility with applications developed under Sun Java System Web Server 7.0

Attributes

The following table describes attributes for the `locale-charset-info` the left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–47 `locale-charset-info` Attributes

Attribute Value	Default Value
default-locale	none

locale-charset-map

Maps a locale to a specific character encoding.

For encodings you can use, see:

`http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html`

Attributes

The following table describes attributes for the `locale-charset-map` element. The left column lists the attribute name, the middle column indicates the default value, and the right column describes what the attribute does.

TABLE A–48 `locale-charset-map` Attributes

Attribute	Default Value	Description
`locale`	none	Specifies the locale name.
`agent`	none	Ignored in Sun Java System Web Server 7.0
`charset`	none	Specifies the character set for that locale.

The following table provides a `locale-charset-map` example, listing the locale and the corresponding charset:

TABLE A–49 `locale-charset-map` Example

Locale	Charset
ja	EUC-JP
zh	UTF-8

message-destination

Specifies the name of a logical message-destination defined within an application. The message-destination-name matches the corresponding message-destination-name in the corresponding Java EE deployment descriptor file.

Subelements

The following table describes subelements for the `message-destination` element

TABLE A–50 `message-destination` Subelements

Elements	Required	Description
"message-destination-name" on page 200	only one	Specifies the name of a logical message destination defined within the corresponding Java EE deployment descriptor file.
"jndi-name" on page 183	only one	Specifies the `jndi-name` of the associated entity

message-destination-name

Specifies the name of a logical message destination defined within the corresponding Java EE deployment descriptor file.

Subelements

none

webservice-description

Specifies a name and optional publish location for a web service.

Subelements

The following table describes subelements for `webservice-description` element

TABLE A–51 `webservice-description` Subelements

Element	Required	Description
"webservice-description-name" on page 200	only one	Specifies a unique name for the web service within a web.
"wsdl-publish-location" on page 201	zero or one	Specifies the URL of a directory to which web services's WSDL is published during deployment.

Attributes

none

webservice-description-name

Specifies a unique name for the web service within a web.

Subelements

none

Attributes

none

wsdl-publish-location

Specifies the URL of a directory to which a web service's WSDL is published during deployment. Any required files are published to this directory, preserving their location relative to the module-specific WSDL directory (`META-INF/wsdl` or `WEB-INF/wsdl`).

Subelements

none

Attributes

none

Alphabetical List of sun-web.xml Elements

This section provides an alphabetical list for the easy lookup of `sun-web.xml` elements.

“cache” on page 184

“cache-helper” on page 186

“cache-helper-ref” on page 189

“cache-mapping” on page 188

“class-loader” on page 193

“constraint-field” on page 192

“context-root” on page 166

“cookie-properties” on page 175

“default-helper” on page 187

“default-resource-principal” on page 182

“description” on page 167

"group-name" on page 169

"http-method" on page 191

"jndi-name" on page 183

"jsp-config" on page 194

"key-field" on page 191

"locale-charset-info" on page 198

"locale-charset-map" on page 199

"manager-properties" on page 171

"message-destination" on page 199

"message-destination-name" on page 200

"`message-security-binding` Element" on page 125

"name" on page 183

"parameter-encoding" on page 197

"password" on page 183

"principal-name" on page 169

"property" on page 166

"refresh-field" on page 190

"res-ref-name" on page 182

"resource-env-ref" on page 176

"resource-env-ref-name" on page 177

"resource-ref" on page 181

"role-name" on page 169

"security-role-mapping" on page 167

"servlet" on page 168

"servlet-name" on page 169

"service-ref" on page 177

“session-config” on page 170

“session-manager” on page 170

“session-properties” on page 174

“store-properties” on page 173

“sun-web-app” on page 163

“timeout” on page 190

“url-pattern” on page 189

“value” on page 193

“webservice-description” on page 200

“webservice-description-name” on page 200

“`webservice-endpoint` Element” on page 124

Note – For a list of `sun-web.xml` elements by category, see “Elements in the sun-web.xml File” on page 162.

Sample Web Application XML Files

This section includes the following:

- “Sample web.xml File” on page 203
- “Sample sun-web.xml File” on page 204

Sample web.xml File

The following is the sample `web.xml` file.

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">-
    <!--

   Copyright 2006 Sun Microsystems, Inc. All rights reserved.
```

```
-->
-
    <web-app>
<display-name>webapps-caching</display-name>
-
    <servlet>
<servlet-name>ServCache</servlet-name>
<servlet-class>samples.webapps.caching.ServCache</servlet-class>
<load-on-startup>0</load-on-startup>
</servlet>
-
    <servlet-mapping>
<servlet-name>ServCache</servlet-name>
<url-pattern>/ServCache</url-pattern>
</servlet-mapping>
-
    <session-config>
<session-timeout>30</session-timeout>
</session-config>
-
    <taglib>
<taglib-uri>/com/sun/web/taglibs/cache</taglib-uri>
<taglib-location>/WEB-INF/sun-web-cache.tld</taglib-location>
</taglib>
</web-app>
```

Sample sun-web.xml File

The following is the sample `sun-web.xml` file.

```
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 8.1
Servlet 2.4//EN" "http://www.sun.com/software/appserver/dtds/sun-web-app_2_4-1.dtd">
 <!--

    Copyright 2006 Sun Microsystems, Inc. All rights reserved.
-->
-
    <sun-web-app>
-
    <session-config>
<session-manager/>
</session-config>
-
    <cache enabled="true" timeout-in-seconds="300">
-
    <cache-mapping>
<servlet-name>ServCache</servlet-name>
```

```
<key-field name="inputtext" scope="request.parameter"/>
-
    <constraint-field name="inputtext" scope="request.parameter">
<value>one</value>
<value>two</value>
</constraint-field>
</cache-mapping>
</cache>
</sun-web-app>
```

Index