



Sun Java™ System

Application Server 7 Developer's Guide to Web Applications

2004Q2

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

Part No: 817-5451

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Use is subject to license terms. This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries.

Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats-Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

L'utilisation est soumise aux termes de la Licence. Cette distribution peut comprendre des composants développés par des tierces parties.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, et le logo Java Coffee Cup sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Ce produit est soumis à la législation américaine en matière de contrôle des exportations et peut être soumis à la réglementation en vigueur dans d'autres pays dans le domaine des exportations et importations. Les utilisations, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers les pays sous embargo américain, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exhaustive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

About This Guide	9
Who Should Use This Guide	9
Using the Documentation	10
How This Guide Is Organized	12
Related Information	13
Documentation Conventions	14
General Conventions	14
Conventions Referring to Directories	15
Contacting Sun	15
Give Us Feedback	16
Obtain Training	16
Contact Product Support	16
 Chapter 1 Web Applications	 19
Introducing Web Applications	20
Servlets	22
JavaServer Pages	22
SHTML	23
CGI	23
Creating a Web Application	23
Deploying a Web Application	24
Debugging a Web Application	24
Internationalization Issues	25
The Server	25
Servlets	25
Servlet Request	25
Servlet Response	26
JSPs	26
Virtual Servers	27
Using the Administration Interface	27

Editing the server.xml File	28
Default Web Modules	28
Servlet and JSP Caching	29
Database Connection Pooling	29
Configuring the Web Container	29
Web Application Examples	29
 Chapter 2 Using Servlets	31
About Servlets	31
Servlet Data Flow	32
Servlet Types	34
Creating Servlets	34
Creating the Class Declaration	35
Overriding Methods	35
Overriding Initialize	36
Overriding Destroy	36
Overriding Service, Get, and Post	37
Accessing Parameters and Storing Data	38
Handling Sessions and Security	39
Accessing Business Logic Components	39
Handling Threading Issues	41
Delivering Client Results	42
Creating a Servlet Response Page	43
Creating a JSP Response Page	43
Invoking Servlets	44
Calling a Servlet with a URL	44
Calling a Servlet Programmatically	45
Servlet Output	46
Using the Administration Interface	46
Editing the server.xml File	47
Caching Servlet Results	47
Caching Features	48
Default Cache Configuration	49
Caching Example	49
CacheHelper Interface	50
CacheKeyGenerator Interface	52
About the Servlet Engine	53
Instantiating and Removing Servlets	53
Request Handling	53
Allocating Servlet Engine Resources	54

Chapter 3 Using JavaServer Pages	55
Introducing JSPs	55
Creating JSPs	56
Designing for Ease of Maintenance	57
Designing for Portability	57
Handling Exceptions	57
JSP Tag Libraries and Standard Portable Tags	57
JSP Caching	58
cache	58
flush	60
Compiling JSPs: The Command-Line Compiler	61
Debugging JSPs	64
 Chapter 4 Creating and Managing User Sessions	 65
Introducing Sessions	65
Sessions and Cookies	66
Sessions and URL Rewriting	66
Sessions and Security	67
Distributed Sessions and Persistence (Enterprise Edition)	67
How to Use Sessions	70
Creating or Accessing a Session	70
Examining Session Properties	71
Binding Data to a Session	72
Invalidating a Session	73
Enabling the Availability Service (Enterprise Edition)	74
Server Instance and Web Container Levels	75
Application and Web Module Levels	76
Web Application Level	77
Session Managers	78
StandardManager	78
Enabling StandardManager	78
Manager Properties for StandardManager	79
PersistentManager	80
Enabling PersistentManager	80
Manager Properties for PersistentManager	82
Store Properties for PersistentManager	83
 Chapter 5 Securing Web Applications	 85
User Authentication by Servlets	85
HTTP Basic Authentication	86
SSL Mutual Authentication	86
Form-Based Login	87
User Authentication for Single Sign-on	87

User Authorization by Servlets	89
Defining Roles	89
Defining Servlet Authorization Constraints	90
Fetching the Client Certificate	90
Security for SHTML and CGI	90
 Chapter 6 Assembling and Deploying Web Modules	93
Web Application Structure	94
Creating Web Deployment Descriptors	95
Deploying Web Applications	96
Using the Command Line	97
Using the Administration Interface	97
Using Sun ONE Studio	98
Deploying to a Cluster (Enterprise Edition)	98
Dynamic Reloading of Web Applications	99
The sun-web-app_2_3-0.dtd File	100
Subelements	100
Data	101
Attributes	101
Elements in the sun-web.xml File	102
General Elements	102
sun-web-app	103
property	104
description	105
Security Elements	105
security-role-mapping	106
servlet	106
servlet-name	107
role-name	107
principal-name	107
group-name	107
Session Elements	108
session-config	108
session-manager	109
manager-properties	110
store-properties	111
session-properties	113
cookie-properties	114
Reference Elements	115
resource-env-ref	115
resource-env-ref-name	116
resource-ref	116
res-ref-name	117

default-resource-principal	117
name	118
password	118
ejb-ref	118
ejb-ref-name	119
jndi-name	119
Caching Elements	119
cache	120
cache-helper	122
default-helper	122
cache-mapping	123
url-pattern	124
cache-helper-ref	125
timeout	125
refresh-field	125
http-method	126
key-field	126
constraint-field	127
value	128
Classloader Elements	128
class-loader	129
JSP Elements	130
jsp-config	130
Internationalization Elements	132
locale-charset-info	132
locale-charset-map	132
parameter-encoding	133
Sample Web Module XML Files	134
Sample web.xml File	134
Sample sun-web.xml File	140
 Chapter 7 Using Server-Parsed HTML	141
Server-Side HTML and J2EE Web Applications	141
Enabling Server-Side HTML	142
Using Server-Side HTML Commands	143
config	144
include	144
echo	145
fsize	145
flastmod	145
exec	145
Environment Variables in Server-Side HTML Commands	146
Embedding Servlets	146

Time Formats	147
Chapter 8 Using CGI	149
CGI and J2EE Web Applications	150
Enabling CGI	150
Specifying CGI Directories	150
Specifying CGI File Extensions	152
Creating Custom Execution Environments for CGI Programs (UNIX only)	153
Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server	155
Specifying a Chroot Directory for a Virtual Server	156
Adding CGI Programs to the Server	159
Setting the Priority of a CGI Program	159
The Query Handler	159
Perl CGI Programs	160
Global CGI Settings	161
CGI Variables	161
Index	165

About This Guide

This guide describes how to create and run Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications that follow the Java™ Servlet and JavaServer Pages™ (JSP™) specifications on the Sun Java™ System Application Server 7, Enterprise Edition. In addition to describing programming concepts and tasks, this guide offers implementation tips and reference material. Topics include results caching, JSP precompilation, session management, security, deployment, SHTML, and CGI.

This preface contains information about the following topics:

- [Who Should Use This Guide](#)
- [Using the Documentation](#)
- [How This Guide Is Organized](#)
- [Related Information](#)
- [Documentation Conventions](#)
- [Contacting Sun](#)

Who Should Use This Guide

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

This guide assumes you are familiar with the following topics:

- J2EE specification
- HTML
- Java programming

- Java APIs as defined in the Java Servlet, JSP, Enterprise JavaBeans™ (EJB™), and Java™ Database Connectivity (JDBC™) specifications
- Structured database query languages such as SQL
- Relational database concepts
- Software development processes, including debugging and source code control

Using the Documentation

The Sun Java System Application Server Standard and Enterprise Edition manuals are available as online files in Portable Document Format (PDF) and Hypertext Markup Language (HTML).

The following table lists tasks and concepts described in the Sun Java System Application Server manuals. The manuals marked *(updated for 7 2004Q2)* have been updated for the Sun Java System Application Server Standard and Enterprise Edition 7 2004Q2 release. The manuals not marked in this way have not been updated since the version 7 Enterprise Edition release.

Table 1 Sun Java System Application Server Documentation Roadmap

For information about	See the following
<i>(Updated for 7 2004Q2)</i> Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of supported hardware, operating system, JDK, and JDBC/RDBMS.	<i>Release Notes</i>
Sun Java System Application Server 7 overview, including the features available with each product edition.	<i>Product Overview</i>
Diagrams and descriptions of server architecture and the benefits of the Sun Java System Application Server architectural approach.	<i>Server Architecture</i>
<i>(Updated for 7 2004Q2)</i> How to get started with the Sun Java System Application Server product. Includes a sample application tutorial. There are two guides, one for Standard Edition and one for Enterprise Edition.	<i>Getting Started Guide</i>
<i>(Updated for 7 2004Q2)</i> Installing the Sun Java System Application Server Standard Edition and Enterprise Edition software and its components, such as sample applications and the Administration interface. For the Enterprise Edition software, instructions are provided for implementing the high-availability configuration.	<i>Installation Guide</i>
<i>(Updated for 7 2004Q2)</i> Evaluating your system needs and enterprise to ensure that you deploy Sun Java System Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying an application server are also discussed.	<i>System Deployment Guide</i>

Table 1 Sun Java System Application Server Documentation Roadmap (*Continued*)

For information about	See the following
Creating and implementing Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications intended to run on the Sun Java System Application Server that follow the open Java standards model for J2EE components such as servlets, Enterprise JavaBeans™ (EJBs™), and JavaServer Pages™ (JSPs™). Includes general information about application design, developer tools, security, assembly, deployment, debugging, and creating lifecycle modules. A comprehensive Sun Java System Application Server glossary is included.	<i>Developer's Guide</i>
(Updated for 7 2004Q2) Creating and implementing J2EE web applications that follow the Java™ Servlet and JavaServer Pages (JSP) specifications on the Sun Java System Application Server. Discusses web application programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include results caching, JSP precompilation, session management, security, deployment, SHTML, and CGI.	<i>Developer's Guide to Web Applications</i>
(Updated for 7 2004Q2) Creating and implementing J2EE applications that follow the open Java standards model for enterprise beans on the Sun Java System Application Server. Discusses Enterprise JavaBeans (EJB) programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include container-managed persistence, read-only beans, and the XML and DTD files associated with enterprise beans.	<i>Developer's Guide to Enterprise JavaBeans Technology</i>
(Updated for 7 2004Q2) Creating Application Client Container (ACC) clients that access J2EE applications on the Sun Java System Application Server.	<i>Developer's Guide to Clients</i>
Creating web services in the Sun Java System Application Server environment.	<i>Developer's Guide to Web Services</i>
(Updated for 7 2004Q2) Java™ Database Connectivity (JDBC™), transaction, Java Naming and Directory Interface™ (JNDI), Java™ Message Service (JMS), and JavaMail™ APIs.	<i>Developer's Guide to J2EE Services and APIs</i>
Creating custom NSAPI plug-ins.	<i>Developer's Guide to NSAPI</i>
(Updated for 7 2004Q2) Information and instructions on the configuration, management, and deployment of the Sun Java System Application Server subsystems and components, from both the Administration interface and the command-line interface. Topics include cluster management, the high-availability database, load balancing, and session persistence. A comprehensive Sun Java System Application Server glossary is included.	<i>Administration Guide</i>
(Updated for 7 2004Q2) Editing Sun Java System Application Server configuration files, such as the <code>server.xml</code> file.	<i>Administrator's Configuration File Reference</i>
Configuring and administering security for the Sun Java System Application Server operational environment. Includes information on general security, certificates, and SSL/TLS encryption. HTTP server-based security is also addressed.	<i>Administrator's Guide to Security</i>

Table 1 Sun Java System Application Server Documentation Roadmap (*Continued*)

For information about	See the following
Configuring and administering service provider implementation for J2EE™ Connector Architecture (CA) connectors for the Sun Java System Application Server. Topics include the Administration Tool, Pooling Monitor, deploying a JCA connector, and sample connectors and sample applications.	<i>J2EE CA Service Provider Implementation Administrator's Guide</i>
(Updated for 7 2004Q2) Migrating your applications to the new Sun Java System Application Server programming model, specifically from iPlanet Application Server 6.x and Sun ONE Application Server 7.0. Includes a sample migration.	<i>Migrating and Redeploying Server Applications Guide</i>
(Updated for 7 2004Q2) How and why to tune your Sun Java System Application Server to improve performance.	<i>Performance Tuning Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server problems.	<i>Troubleshooting Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server error messages.	<i>Error Message Reference</i>
(Updated for 7 2004Q2) Utility commands available with the Sun Java System Application Server; written in manpage style.	<i>Utility Reference Manual</i>
Using the Sun™ Java System Message Queue 3.5 software.	The Sun Java System Message Queue documentation at: http://docs.sun.com/db?p=prod/sl.s1msgqu

How This Guide Is Organized

This guide provides a Sun Java System Application Server environment overview for designing web applications. The content is as follows:

- [Chapter 1, “Web Applications”](#)
This chapter introduces web applications and describes how they are supported in Sun Java System Application Server.
- [Chapter 2, “Using Servlets”](#)
This chapter describes how to create and use servlets.
- [Chapter 3, “Using JavaServer Pages”](#)
This chapter describes how to create and use JavaServer Pages (JSPs).
- [Chapter 4, “Creating and Managing User Sessions”](#)

This chapter describes how to create and manage a session that allows users and transaction information to persist between interactions.

- [Chapter 5, “Securing Web Applications”](#)

This chapter describes how to write a secure web application for the Sun Java System Application Server.

- [Chapter 6, “Assembling and Deploying Web Modules”](#)

This chapter describes how web modules are assembled and deployed in Sun Java System Application Server.

- [Chapter 7, “Using Server-Parsed HTML”](#)

This chapter describes how to use server-parsed HTML with the Sun Java System Application Server.

- [Chapter 8, “Using CGI”](#)

This chapter describes how to use CGI with the Sun Java System Application Server.

Finally, an [Index](#) is provided.

Related Information

You can find a directory of URLs for the official specifications at *install_dir/docs/index.htm*. Additionally, the following resources may be useful:

Programming with Servlets and JSPs:

Java Servlet Programming, by Jason Hunter, O'Reilly Publishing

Java Threads, 2nd Edition, by Scott Oaks & Henry Wong, O'Reilly Publishing

Programming with JDBC:

Database Programming with JDBC and Java, by George Reese, O'Reilly Publishing

JDBC Database Access With Java: A Tutorial and Annotated Reference (Java Series), by Graham Hamilton, Rick Cattell, & Maydene Fisher

Documentation Conventions

This section describes the types of conventions used throughout this guide:

- [General Conventions](#)
- [Conventions Referring to Directories](#)

General Conventions

The following general conventions are used in this guide:

- **File and directory paths** are given in UNIX® format (with forward slashes separating directory names). For Windows versions, the directory paths are the same, except that backslashes are used to separate directories.

- **URLs** are given in the format:

`http://server.domain/path/file.html`

In these URLs, *server* is the server name where applications are run; *domain* is your Internet domain name; *path* is the server's directory structure; and *file* is an individual filename. Italic items in URLs are placeholders.

- **Font conventions** include:
 - The monospace font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
 - *Italic* type is used for code variables.
 - *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
 - **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.
- **Installation root directories** for most platforms are indicated by *install_dir* in this document. Exceptions are noted in [“Conventions Referring to Directories” on page 15](#).

By default, the location of *install_dir* on **most** platforms is:

- Solaris and Linux file-based installations:

user's home directory/sun/appserver7

- Windows, all installations:

system drive: \Sun\AppServer7

For the platforms listed above, *default_config_dir* and *install_config_dir* are identical to *install_dir*. See “[Conventions Referring to Directories](#)” on page 15 for exceptions and additional information.

- **Instance root directories** are indicated by *instance_dir* in this document, which is an abbreviation for the following:

default_config_dir/domains/domain/instance

- **UNIX-specific descriptions** throughout this manual apply to the Linux operating system as well, except where Linux is specifically mentioned.

Conventions Referring to Directories

By default, when using the Solaris package-based or Linux RPM-based installation, the application server files are spread across several root directories. This guide uses the following document conventions to correspond to the various default installation directories provided:

- *install_dir* refers to /opt/SUNWappserver7, which contains the static portion of the installation image. All utilities, executables, and libraries that make up the application server reside in this location.
- *default_config_dir* refers to /var/opt/SUNWappserver7/domains, which is the default location for any domains that are created.
- *install_config_dir* refers to /etc/opt/SUNWappserver7/config, which contains installation-wide configuration information such as licenses and the master list of administrative domains configured for this installation.

Contacting Sun

You might want to contact Sun Microsystems in order to:

- [Give Us Feedback](#)
- [Obtain Training](#)
- [Contact Product Support](#)

Give Us Feedback

If you have general feedback on the product or documentation, please send this to:

<http://www.sun.com/hwdocs/feedback/>

Obtain Training

Application Server training courses are available at:

http://training.sun.com/US/catalog/enterprise/web_application.html/

Visit this site often for new course availability on the Sun Java System Application Server.

Contact Product Support

If you have problems with your system, contact customer support using one of the following mechanisms:

- The online support web site at:
<http://www.sun.com/supporttraining/>
- The telephone dispatch number associated with your maintenance contract

Please have the following information available prior to contacting support. This helps to ensure that our support staff can best assist you in resolving problems:

- Description of the problem, including the situation where the problem occurs and its impact on your operation
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem. Here are some of the commonly used commands:
 - **Solaris:** `pkginfo, showrev`
 - **Linux:** `rpm`
 - **All:** `asadmin version --verbose`
- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps
- Configuration files such as:
 - `instance_dir/config/server.xml`

- a web application's `web.xml` file,
when a web application is involved in the problem
- For an application, whether the problem appears when it is running in a cluster or standalone

Contacting Sun

Web Applications

This chapter describes how web applications are supported in the Sun Java™ System Application Server 7, Enterprise Edition, and includes the following sections:

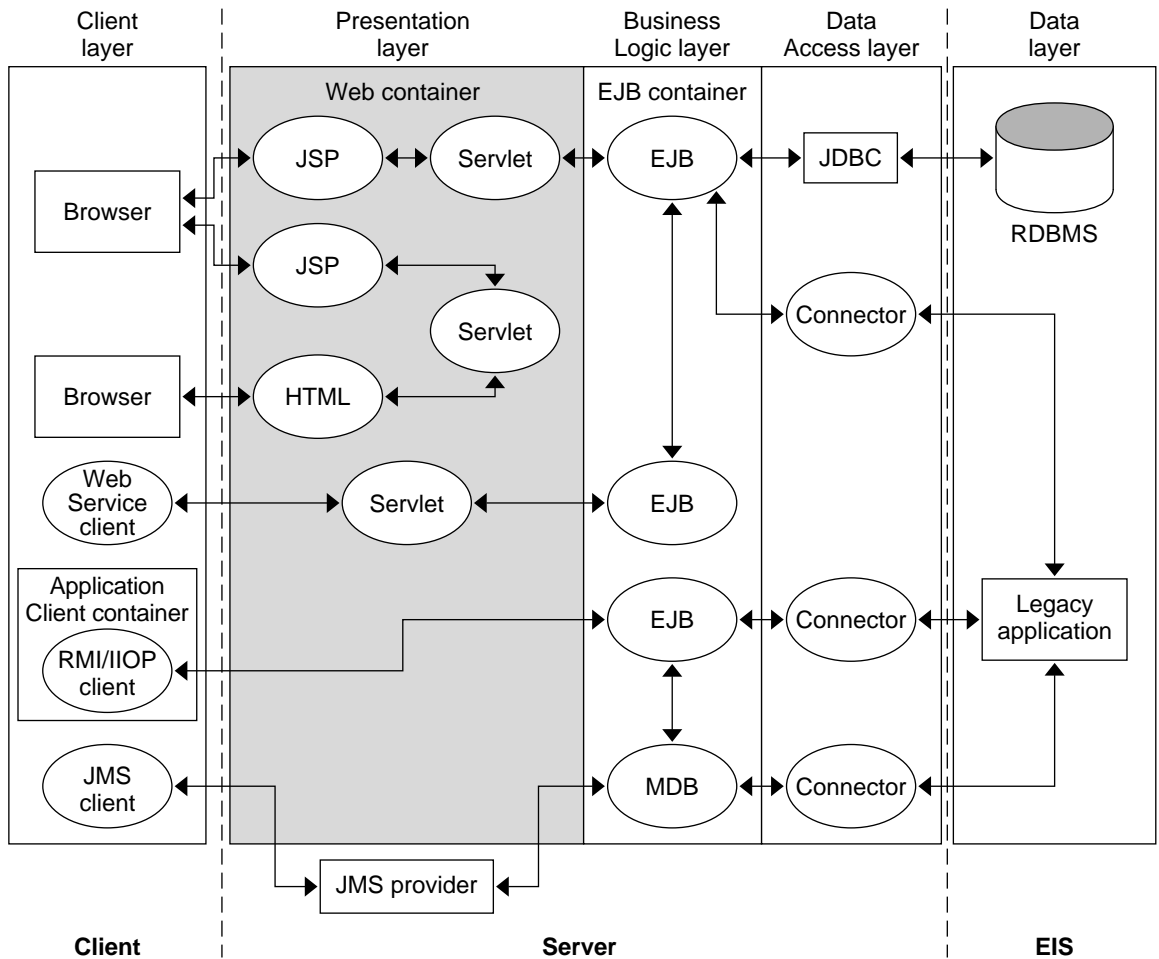
- [Introducing Web Applications](#)
- [Creating a Web Application](#)
- [Deploying a Web Application](#)
- [Debugging a Web Application](#)
- [Internationalization Issues](#)
- [Virtual Servers](#)
- [Default Web Modules](#)
- [Servlet and JSP Caching](#)
- [Database Connection Pooling](#)
- [Configuring the Web Container](#)
- [Web Application Examples](#)

Introducing Web Applications

Sun Java System Application Server 7 supports the Java™ Servlet 2.3 API specification, which allows servlets and JavaServer Pages™ (JSP™) pages to be included in web applications.

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

The following figure illustrates details of the Java™ 2 Platform, Enterprise Edition (J2EE™ platform) environment. Web applications are in the presentation layer. Two web applications are running in the web container: one of them is part of a full application that includes Enterprise JavaBeans™ (EJB™) modules and connectors, while the other is an individually deployed web module.

Figure 1-1 Web applications in the J2EE environment

In addition, Sun Java System Application Server 7 supports SHTML and CGI, which are non-J2EE application components.

This section includes summaries of the following topics:

- [Servlets](#)
- [JavaServer Pages](#)
- [SHTML](#)
- [CGI](#)

Servlets

Java servlets are server-side Java programs that application servers can run to 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 Application Server 7 supports the Java Servlet Specification version 2.3.

NOTE	Servlet API version 2.3 is fully backward compatible with versions 2.1 and 2.2, so all existing servlets will continue to work without modification or recompilation.
-------------	---

To develop servlets, use Sun Microsystems' Java Servlet API. For information about using the Java Servlet API, see the documentation provided by Sun Microsystems at:

<http://java.sun.com/products/servlet/index.html>

For information about developing servlets in Sun Java System Application Server, see [Chapter 2, "Using Servlets."](#)

JavaServer Pages

Sun Java System Application Server 7 supports JavaServer Pages (JSP) Specification version 1.2.

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.

One of the main benefits of JSPs is that they are like HTML pages. The web page designer simply writes a page that uses HTML and JSP tags and puts it on their application server. The page is compiled automatically when it is deployed. What the web page designer needs to know about Java classes and Java compilers is minimal.

Sun Java System Application Server supports precompilation of JSPs, however, and this is especially useful for production servers.

JSP pages can access full Java functionality in the following ways:

- by embedding Java code directly in scriptlets in the page

- by accessing Java beans
- by using server-side tags that include Java servlets

Both beans and servlets are Java classes that need to be compiled, but they can be defined and compiled by a Java programmer, who then publishes the interface to the bean or the servlet. The web page designer can access a pre-compiled bean or servlet from a JSP page.

Sun Java System Application Server 7 supports JSP tag libraries and standard portable tags.

For information about creating JSPs, see Sun Microsystem's JavaServer Pages web site at:

<http://java.sun.com/products/jsp/index.html>

For information about Java Beans, see Sun Microsystem's JavaBeans web page at:

<http://java.sun.com/beans/index.html>

For information about developing JSPs in Sun Java System Application Server, see [Chapter 3, "Using JavaServer Pages."](#)

SHTML

HTML files can contain tags that are executed on the server. In addition to supporting the standard server-side tags, or SSI's, Sun Java System Application Server 7 allows you to embed servlets and define your own server-side tags. For more information, see [Chapter 7, "Using Server-Parsed HTML."](#)

CGI

Common Gateway Interface (CGI) programs run on the server and generate a response to return to the requesting client. CGI programs can be written in various languages, including C, C++, Java, Perl, and as shell scripts. CGI programs are invoked through URL invocation. Sun Java System Application Server complies with the version 1.1 CGI specification. For more information, see [Chapter 8, "Using CGI."](#)

Creating a Web Application

To create a web application:

1. Create a directory for all 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 3, “Using JavaServer Pages.”](#)
4. Create any needed servlets. For more information, see [Chapter 2, “Using Servlets.”](#)
5. Compile the servlets. For details about precompiling JSPs, see [“Compiling JSPs: The Command-Line Compiler” on page 61.](#)
6. Organize the web application as described in [“Web Application Structure” on page 94.](#)
7. Create the deployment descriptor files. For more information, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)
8. Package the web application in a WAR file if desired. This is optional. For example:

```
jar -cvf module_name.war *
```
9. Deploy the web application. For more information, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

You can create a web application by hand, or you can use Sun ONE Studio 5. For more information about Sun ONE Studio, see the *Sun ONE Studio 5, Enterprise Edition Tutorial*.

Deploying a Web Application

Web application deployment descriptor files are created by the Sun Java System Application Server Administration interface during deployment. You can also create these by hand. These descriptor files are packaged within Web Application Archive (.war) files. They contain metadata, plus information that identifies the servlet or JSP and establishes its application role. For more information about these descriptor files, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Debugging a Web Application

For information about debugging applications, see the *Sun Java System Application Server Developer's Guide*.

Internationalization Issues

This section covers internationalization as it applies to the following:

- [The Server](#)
- [Servlets](#)
- [JSPs](#)

The Server

To set the default locale of the entire Sun Java System Application Server, which determines the locale of the Administration interface, the logs, and so on, do one of the following:

- Go to the server instance page of the Administration interface, click on the Advanced tab, type a value in the Locale field, click on the Save button, click on the General tab, and select the Apply Changes button.
- Set the `locale` attribute of the `server` element in the `server.xml` file, then restart the server. For more information about this file, see the *Sun Java System Application Server Administrator's Configuration File Reference*.

Servlets

This section explains how the Sun Java System Application Server determines the character encoding for the servlet request and the servlet response.

For encodings you can use, see:

<http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html>

Servlet Request

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

- The `setCharacterEncoding()` method.
- A hidden field in the form, specified by the `parameter-encoding` element in the `sun-web.xml` file. For more information, see [“parameter-encoding” on page 133](#).

- The character encoding set in the `locale-charset-info` element in the `sun-web.xml` file. For more information about this element, see [“Internationalization Elements” on page 132](#).
- The default, which is `ISO-8859-1`.

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 `setContentType()` method.
- The `setLocale()` method.
- The default, which is `ISO-8859-1`.

To send non-ASCII characters in the response headers, set the `useResponseCTForHeaders` property, which is described in the section [“sun-web-app” on page 103](#).

JSPs

For encodings you can use, see:

<http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html>

To set the character encoding of a JSP, use the `page` directive. For example:

```
<%@ page contentType="text/html; charset=Shift_JIS" %>
```

The `contentType` attribute defines the following:

- The character encoding for the JSP page.
- The character encoding for the response of the JSP page.
- The MIME type for the response of the JSP page.

The default value is `text/html; charset=ISO-8859-1`.

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

- The `page` directive and `contentType` attribute of JSP file.
- The default, which is `ISO-8859-1`.

Some JSP pages can deliver content using different content types and character encodings depending on the request time input. Dynamic setting of content type relies on an underlying invocation of `response.setContentType()`. This method can be invoked as long as no content has been sent to the response stream.

Virtual Servers

A virtual server, also called a virtual host, is a virtual web server that serves content targeted for a specific URL. Multiple virtual servers may serve content using the same or different host names, port numbers, or IP addresses. The HTTP service can direct incoming web requests to different virtual servers based on the URL.

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

Web applications and J2EE applications containing web components can be assigned to virtual servers. You can assign virtual servers in either of these ways:

- [Using the Administration Interface](#)
- [Editing the server.xml File](#)

Using the Administration Interface

You can assign a virtual server to a web module during deployment as described in [“Deploying Web Applications” on page 96](#). To use the Administration interface to specify a virtual server’s default web module:

1. Deploy the web application or J2EE application as described in [“Deploying Web Applications” on page 96](#).
2. Open the HTTP Server component under your server instance.
3. Open the Virtual Servers component under the HTTP Server component.
4. Select the virtual server to which you want to assign the web application.
5. Select the web application from the Default Web Module drop-down list.
6. Select the Save button.
7. Go to the server instance page and select the Apply Changes button.

For more information, see [“Default Web Modules” on page 28](#).

Editing the server.xml File

When a web module is deployed as part of an application, a `j2ee-application` element is created for it in `server.xml` during deployment. When a web module is deployed as an individual module, a `web-module` element is created for it in `server.xml` during deployment. The `j2ee-application` and `web-module` elements both have a `virtual-servers` attribute, which specifies a list of virtual server IDs. The `virtual-servers` attribute is empty by default, which means that the web application is assigned to all virtual servers.

Each `virtual-server` element in `server.xml` has a `default-web-module` attribute, which allows you to configure a default web module for each virtual server. A default web module for the default virtual server is provided at installation. For more information, see [“Default Web Modules” on page 28](#).

For more information about `server.xml` and virtual servers, see the Sun Java System *Application Server Administrator’s Configuration File Reference*.

Default Web Modules

You can assign a default web module to the default virtual server and to each new virtual server you create. For details, see [“Virtual Servers” on page 27](#). To access the default web module for a virtual server, point your browser to the URL for the virtual server, but do not supply a context root. For example:

```
http://myvserver:3184/
```

A virtual server with no default web module assigned serves HTML or JSP content from its document root, which is usually `instance_dir/docroot`. To access this HTML or JSP content, point your browser to the URL for the virtual server, do not supply a context root, but specify the target file.

For example:

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

Servlet and JSP Caching

The Sun Java System Application Server has the ability to cache servlet or JSP results in order to make subsequent calls to the same servlet or JSP faster. The Sun Java System Application Server caches the request results for a specific amount of time. In this way, if another data call occurs the Sun Java System Application Server 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.

For more information about response caching as it pertains to servlets, see [“Caching Servlet Results” on page 47](#). For more information about JSP caching, see [“JSP Caching” on page 58](#).

Database Connection Pooling

Database connection pooling enhances the performance of servlet or JSP database interactions. There are several Java™ Database Connectivity (JDBC™) 2.0 compatible drivers that support connection pooling, for example Pointbase, Oracle 8i update, and CloudScape 3.0. For more information about JDBC, see the *Sun Java System Application Server Developer's Guide to J2EE Services and APIs*.

Configuring the Web Container

You can configure logging in the web container for the entire server in these ways:

- By using the Administration interface; see the *Sun Java System Application Server Administrator's Guide*.
- By editing the `server.xml` file; see the *Sun Java System Application Server Administrator's Configuration File Reference*.

Web Application Examples

Sample web applications that you can examine and deploy are included in Sun Java System Application Server, in the `install_dir/samples/webapps` directory. Each sample has its own documentation.

Using Servlets

This chapter describes how to create effective servlets to control application interactions running on a Sun Java System Application Server, including standard servlets. In addition, this chapter describes the Sun Java System Application Server features to use to augment the standards.

This chapter contains the following sections:

- [About Servlets](#)
- [Creating Servlets](#)
- [Invoking Servlets](#)
- [Servlet Output](#)
- [Caching Servlet Results](#)
- [About the Servlet Engine](#)

About Servlets

Servlets, like applets, are reusable Java applications. However, servlets run on an application server or web server rather than in a web browser.

Servlets supported by the Sun Java System Application Server are based on the Java Servlet Specification v2.3. All relevant specifications are accessible from *install_dir/docs/index.htm*, where *install_dir* is the directory where the Sun Java System Application Server is installed.

Servlets are used for an application's presentation logic. A servlet acts as an application's central dispatcher by processing form input, invoking business logic components encapsulated in EJB components, and formatting web page output using JSPs. Servlets control the application flow from one user interaction to the next by generating content in response to user requests.

The fundamental characteristics are:

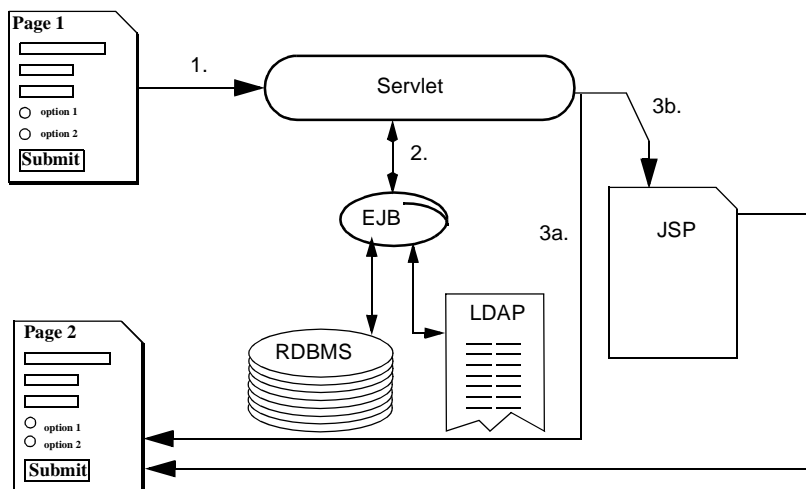
- Servlets are created and managed at runtime by the Sun Java System Application Server servlet engine.
- Servlets operate on input data that is encapsulated in a `request` object.
- Servlets respond to a query with data encapsulated in a `response` object.
- Servlets call EJB components to perform business logic functions.
- Servlets call JSPs to perform page layout functions.
- Servlets are extensible; use the APIs provided with the Sun Java System Application Server to add functionality.
- Servlets provide user session information persistence between interactions.
- Servlets can be part of an application or they can reside discretely on the application server so they are available to multiple applications.
- Servlets can be dynamically reloaded while the server is running.
- Servlets are addressable with URLs; buttons on an application's pages often point to servlets.
- Servlets can call other servlets.

Servlet Data Flow

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 orchestrates a response by generating content, often through business logic components, which are EJB components. 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.

The following illustration shows the information flow to and from the servlet.

Figure 2-1 Servlet Data Flow Steps



The steps are as follows:

1. Servlet processes the client request
2. Servlet generates content
3. Servlet creates response and either:
 - a. Sends it back directly to the client
 - or
 - b. Dispatches the task to a JSP

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

Servlet Types

There are two main servlet types:

- 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 Application Server environment.

For both servlet types, implement the constructor method `init()` and the destructor method `destroy()` to initialize or deallocate resources, respectively.

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, perform the following tasks:

- Design the servlet into your application, or, if accessed in a generic way, design it to access no application data.
- Create a class that extends either `GenericServlet` or `HttpServlet`, overriding the appropriate methods so it handles requests.
- Use the Sun Java System Application Server Administration interface to create a web application deployment descriptor. For details, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

The rest of this section discusses the following topics:

- [Creating the Class Declaration](#)
- [Overriding Methods](#)

- [Accessing Parameters and Storing Data](#)
- [Handling Sessions and Security](#)
- [Accessing Business Logic Components](#)
- [Handling Threading Issues](#)
- [Delivering Client Results](#)

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`. Since Sun Java System Application Server servlets exist in an HTTP environment, you should extend `HttpServlet`. 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 servlet 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 `doOperation()` methods for HTTP servlets. This method accepts incoming requests, processing them according to the instructions you provide, and directs the output appropriately. You can create other methods in a servlet as well.

Business logic may involve database access to perform a transaction or passing the request to an EJB component.

Overriding Initialize

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 All `init()` methods must call `super.init(ServletConfig)` to set their scope. This makes the servlet's configuration object available to other servlet methods. If this call is omitted, a 500 `SC_INTERNAL_SERVER_ERROR` appears in the browser when the servlet starts up.

NOTE A web application is not started if any of its components, such as a filter, throws a `ServletException` during initialization. This is to ensure that if any part of the web application runs, all of it runs. It is especially important that a web application fail if security components fail.

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 life cycle. Resources should be appropriately closed and dereferenced 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 Initialize”](#) above:

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

Overriding Service, Get, and Post

When a request is made, the Sun Java System Application Server hands the incoming data to the servlet engine to process the 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, GET, and so on). For example, HTTP POST requests are routed to the `doPost()` method, HTTP GET requests are routed to the `doGet()` method, and so on. This enables the servlet to perform different request data processing depending on the transfer method. Since the routing takes place in `service()`, there is no need to generally override `service()` in an HTTP servlet. Instead, override `doGet()`, `doPost()`, and so on, depending on the expected request type.

The automatic routing in an HTTP servlet is based simply on a call to `request.getMethod()`, which provides the HTTP transfer method. In a Sun Java System Application Server, request data is already preprocessed into a name-value list by the time the servlet sees the data, so simply overriding the `service()` method in an HTTP servlet does not lose any functionality. However, this does make the servlet less portable, since it is now dependent on preprocessed request data.

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 accessing EJB components to perform business transactions, 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()`. Note that implementing both methods to provide for both input types or simply pass the request object to a central processing method.

For 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 `doOperation()` method, including session management, user authentication, dispatching EJB components and JSPs, and accessing Sun Java System Application Server features.

If a servlet intends to call the `RequestDispatcher` method `include()` or `forward()`, be aware the request information is no longer sent as HTTP POST, GET, and so on. In other words, if a servlet overrides `doPost()`, it may not process anything if another servlet calls it, 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 45](#).

NOTE	Arbitrary binary data, such as uploaded files or images, can be problematic, since the web connector translates incoming data into name-value pairs by default. You can program the web connector to properly handle these kinds of data and package them correctly in the request object.
-------------	--

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");
```

This shows one way to transfer data to a JSP, since JSPs have access to the request object as an implicit bean.

Handling Sessions and Security

From a web or application 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 their last interaction were 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 4, “Creating and Managing User Sessions.”](#)

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 whatever additional information to retain. Application components can then query the session object to obtain user authentication.

To provide a secure user session for your application, see [Chapter 5, “Securing Web Applications.”](#)

Accessing Business Logic Components

In the Sun Java System Application Server programming model, you implement business logic, including database or directory transactions and complex calculations, in EJB components. A request object reference can be passed as an EJB parameter to perform the specified task.

Store the results from database transactions in `JDBC ResultSet` objects and pass object references to other components for formatting and delivery to the client. Also, store request object results by using the `request.setAttribute()` method, or in the session by using the `session.setAttribute()` method. Objects stored in the request object are valid only for the request length, or in other words for this particular servlet thread. Objects stored in the session persist for the session duration, which can span many user interactions.

This example shows a servlet accessing an EJB component called `ShoppingCart`. The servlet creates a cart handle by casting the user's session ID as a cart after importing the cart's remote interface. The cart is stored in the user's session.

```
import cart.ShoppingCart;

// Get the user's session and shopping cart
HttpSession session = request.getSession(true);
```

```

ShoppingCart cart =
    (ShoppingCart)session.getAttribute(session.getId());

// If the user has no cart, create a new one
if (cart == null) {
    String jndiNm = "java:comp/env/ejb/ShoppingCart";
    javax.naming.Context initCtx = null;
    Object home = null;
    try {
        initCtx = new javax.naming.InitialContext(env);
        java.util.Properties props = null;
        home = initCtx.lookup(jndiNm);
        cart = ((IShoppingCartHome) home).create();
        session.setValue(session.getId(), cart);
    }
    catch (Exception ex) {
        .....
        .....
    }
}

```

Access EJB components from servlets by using the Java Naming and Directory Interface™ (JNDI) to establish a handle, or proxy, to the EJB component. Next, refer to the EJB component as a regular object; overhead is managed by the bean's container.

This example shows JNDI looking up a proxy for the shopping cart:

```

String jndiNm = "java:comp/env/ejb/ShoppingCart";
javax.naming.Context initCtx;
Object home;
try
{
    initCtx = new javax.naming.InitialContext(env);
}
catch (Exception ex)
{
    return null;
}
try
{
    java.util.Properties props = null;
    home = initCtx.lookup(jndiNm);
}
catch (javax.naming.NameNotFoundException e)
{
    return null;
}

```



```

    }
    catch(javax.naming.NamingException e)
    {
        return null;
    }
    try
    {
        IShoppingCart cart = ((IShoppingCartHome) home).create();
    }
    catch (...) {...}

```

For more information on EJB components, see the *Sun Java System Application Server Developer's Guide to Enterprise JavaBeans Technology*.

NOTE To avoid collisions with names of other enterprise resources in JNDI, and to avoid portability problems, all names in a Sun Java System Application Server application should begin with the string `java:comp/env`.

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 how 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.

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 Application Server, 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 Application Server specific web application deployment descriptor. For more information on the Sun Java System Application Server web application deployment descriptor, see [Chapter 6, “Assembling and Deploying Web Modules.”](#) A single-threaded servlet is slower under load because new requests must wait for a free instance in order to proceed, but 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...
}

```

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:

- [Creating a Servlet Response Page](#)
- [Creating a JSP Response Page](#)

Creating a Servlet Response Page

Generate the output page within a servlet by writing to the output stream. The best 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 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 hands the JSP interaction control. The servlet is no longer involved with the current interaction's 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. For more information about this method, see the Java Servlet Specification, v2.3, section 8.

For more information about JSPs, see [Chapter 3, “Using JavaServer Pages.”](#)

Invoking Servlets

You can invoke a servlet by directly addressing it from an application page with a URL or calling it programmatically from an already running servlet. See the following sections:

- [Calling a Servlet with a URL](#)
- [Calling a Servlet Programmatically](#)

Calling a Servlet with a URL

You can call servlets by using URLs embedded as links in an application’s 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.

Table 2-1 URL Fields for Servlets Within an Application	
URL element	Description
<i>server:port</i>	The IP address (or host name) and optional port number. To access the default web module for a virtual server, specify only this URL section. You do not need to specify the <i>context_root</i> or <i>servlet_name</i> unless you also wish to specify name-value parameters.

Table 2-1 URL Fields for Servlets Within an Application (*Continued*)

URL element	Description
<i>context_root</i>	For an application, the context root is defined in the <code>context-root</code> element of the <code>application.xml</code> or <code>sun-application.xml</code> file. For an individually deployed web module, you specify the context root during deployment.
<i>servlet</i>	Only needed if no <code>servlet-mapping</code> is defined in the <code>web.xml</code> file.
<i>servlet_name</i>	The <code>servlet-name</code> (or <code>servlet-mapping</code> if defined) as configured in the <code>web.xml</code> file.
<i>?name=value...</i>	Optional servlet name-value parameters.

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

```
http://www.sun.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/OfficeApp/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("/ShowSupplies");
dispatcher.include(request, response);
```

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

NOTE Forwarding a request means the original servlet is no longer involved with the current interaction output after `forward()` is invoked. Therefore, only one `forward()` call can be made in a particular servlet.

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

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

Servlet Output

`ServletContext.log` messages are sent to the server log.

By default, the `System.out` and `System.err` output of servlets are sent to the server log, and during start-up server log messages are echoed to the `System.err` output. You can change these defaults in these ways:

- [Using the Administration Interface](#)
- [Editing the server.xml File](#)

Using the Administration Interface

Use the Administration interface as follows:

1. Click on the Logging tab of the server instance page in the Administration interface.
2. Check or uncheck these boxes:
 - Log stdout content to event log - If true, `System.out` output is sent to the server log.
 - Log stderr content to event log - If true, `System.err` output is sent to the server log.
 - Echo to stderr - If true, server log messages are echoed `System.err`.
3. Click on the Save button.
4. Go to the server instance page and select the Apply Changes button.

For more information, see the *Sun Java System Application Server Administrator's Guide*.

Editing the server.xml File

Edit the `server.xml` file as follows, then restart the server:

```
<log-service    log-stdout=false  
                log-stderr=false  
                echo-log-messages-to-stderr=false
```

For more information about `server.xml`, see the *Sun Java System Application Server Administrator's Configuration File Reference*.

Caching Servlet Results

The Sun Java System Application Server 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 Application Server caches the request results for a specific amount of time. In this way, if another data call occurs, the Sun Java System Application Server 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.

Whether to cache results and how to cache them depends on the data involved. For example, it makes no sense 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 Application Server 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 Application Server feature.

For more information about JSP caching, see [“JSP Caching” on page 58](#).

NOTE	For information about caching static file content, see the <code>nsfc.conf</code> file, described in the <i>Sun Java System Application Server Administrator's Configuration File Reference</i> .
-------------	---

The rest of this section covers the following topics:

- [Caching Features](#)
- [Default Cache Configuration](#)

- [Caching Example](#)
- [CacheHelper Interface](#)
- [CacheKeyGenerator Interface](#)

Caching Features

Sun Java System Application Server 7 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, this includes 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 50](#).
- You can determine cache key generation programmatically by writing a cache key generator class. See [“CacheKeyGenerator Interface” on page 52](#).
- 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.
- Since 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 evict cache entries if the maximum cache size is exceeded.
- Key generation concatenates the servlet path with key field values, if any are specified.

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 'lilies' or 'roses'
       -- but no other catalog varieties:
       -- /orchard/catalog?best&category='lilies'
       -- /orchard/catalog?best&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 doesn't need to be present in all the requests -->
    <constraint-field name='SKUnum' scope='request.parameter'>
        <value match-expr='in-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 119](#).

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 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
 * don't 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;
}

```

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 122](#).

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 HttpServletRequest
     *  @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);
}

```

About the Servlet Engine

Servlets exist in and are managed by the servlet engine in the Sun Java System Application Server. The servlet engine is an internal object that handles all servlet meta functions. These functions include instantiation, initialization, destruction, access from other components, and configuration management.

Instantiating and Removing Servlets

After the servlet engine instantiates the servlet, the servlet engine runs its `init()` method to perform any necessary initialization. Override this method to perform an initialize a function for the servlet's life, such as initializing a counter.

When a servlet is removed from service, the server engine calls the `destroy()` method in the servlet so that the servlet can perform any final tasks and deallocate resources. Override this method to write log messages or clean up any lingering connections that won't be caught in garbage collection.

Request Handling

When a request is made, the Sun Java System Application Server hands the incoming data to the servlet engine. The servlet engine processes the request's input data, such as form data, cookies, session information, and URL name-value pairs, into an `HttpServletRequest` request object type.

The servlet engine also creates an `HttpServletResponse` response object type. The engine then passes both as parameters to the servlet's `service()` method.

In an HTTP servlet, the default `service()` method routes requests to another method based on an HTTP transfer method, such as POST, GET, and so on. For example, HTTP POST requests are sent to the `doPost()` method, HTTP GET requests are sent to the `doGet()` method, and so on. This enables the servlet to process request data differently, depending on which transfer method is used. Since the routing takes place in the service method, you generally do not override `service()` in an HTTP servlet. Instead, override `doGet()`, `doPost()`, and so on, depending on the request type you expect.

TIP To enable automatic routing in an HTTP servlet, call `request.getMethod()`, which provides the HTTP transfer method. Since request data is already preprocessed into a name value list in the Sun Java System Application Server, you could simply override the `service()` method in an HTTP servlet without losing functionality. However, this does make the servlet less portable, since it is now dependent on preprocessed request data.

To perform the tasks to answer a request, override the `service()` method for generic servlets, and the `doGet()` or `doPost()` methods for HTTP servlets. Very often, this means accessing EJB components to perform business transactions, collating the information in the request object or in a `JDBC ResultSet` object, and then passing the newly generated content to a JSP for formatting and delivery back to the user.

Allocating Servlet Engine Resources

By default, the servlet engine creates a thread for each new request. This is less resource intensive than instantiating a new servlet copy in memory for each request. Avoid threading issues, since each thread operates in the same memory space where servlet object variables can overwrite each other.

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.

For more information about the `sun-web.xml` file, see [Chapter 6, “Assembling and Deploying Web Modules.”](#) For more information on threading issues, see [“Handling Threading Issues”](#) on page 41.

Using JavaServer Pages

This chapter describes how to use JavaServer Pages (JSPs) as page templates in a Sun Java System Application Server web application.

This chapter contains the following sections:

- [Introducing JSPs](#)
- [Creating JSPs](#)
- [JSP Tag Libraries and Standard Portable Tags](#)
- [JSP Caching](#)
- [Compiling JSPs: The Command-Line Compiler](#)
- [Debugging JSPs](#)

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 Application Server are based on the JSP 1.2 specification. This specification is accessible from *install_dir/docs/index.htm*; *install_dir* is where the Sun Java System Application Server is installed.

In a Sun Java System Application Server application, JSPs are the individual pages that make up an 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.

The Sun Java System Application Server 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 server's JSP engine, which is responsible for interpreting JSP specific tags 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 create JSPs in 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 JSP-specific tags into the raw source code where needed, and by giving the file a `.jsp` extension.

JSPs that adhere to the JSP 1.2 specification follow XML syntax for the most part, which is consistent with HTML. For a summary of the JSP tags you can use, see [“JSP Tag Libraries and Standard Portable Tags” on page 57](#).

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 by hand. 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 2, “Using Servlets.”](#)

Here are a few additional JSP design tips:

- [Designing for Ease of Maintenance](#)
- [Designing for Portability](#)
- [Handling Exceptions](#)

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 JSP is compiled into a servlet or when a request arrives.

Designing for Portability

JSPs can be completely portable between different applications and different servers. A disadvantage is that they have no particular application data knowledge, but this is only a problem if they require that kind of data.

One possible use for 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 tags. A slightly less minimal JSP might 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 Application Server generates an exception, usually a 404 or 500 error. To avoid this problem, set the `errorPage` attribute of the `<%@ page %>` tag.

JSP Tag Libraries and Standard Portable Tags

Sun Java System Application Server supports tag libraries and standard portable tags. For more information about tag libraries, see the JSP 1.2 specification at:

<http://java.sun.com/products/jsp/download.html>

JSP Caching

JSP caching lets you 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, weather information, and so on. 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 47](#).

JSP caching uses the custom tag library support provided by JSP 1.2. JSP caching is implemented by a tag library packaged into the `install_dir/lib/appserv-tags.jar` file, which you can copy into the `WEB-INF/lib` directory of your web application. The `appserv-tags.tld` tag description file is in this JAR file and in the `install_dir/lib/tlds` directory.

You refer to these tags in your JSP files as follows:

```
<%@ taglib prefix="prefix" uri="Sun Java System Application Server Tags" %>
```

Subsequently, the cache tags are available as `<prefix:cache>` and `<prefix:flush>`. For example, if your *prefix* is *mypfx*, the cache tags are available as `<mypfx:cache>` and `<mypfx:flush>`.

If you wish to use a different URI for this tag library, you can use an explicit `<taglib>` element in your `web.xml` file.

The tags are as follows:

- [cache](#)
- [flush](#)

cache

The cache tag 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.

Table 3-1 cache Attributes

Attribute	Default	Description
key	<i>ServletPath_Suffix</i>	(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: <i>s</i> for seconds, <i>m</i> for minutes, <i>h</i> for hours, <i>d</i> for days. For example, 2h specifies two hours.
nocache	false	(optional) If set to <i>true</i> , the body content is executed and served as if there were no <i>cache</i> tag. This offers a way to programmatically decide whether the cached response should be sent or whether the body has to be executed, though the response is not cached.
refresh	false	(optional) If set to <i>true</i> , the body content is executed and the response is cached again. This lets you programmatically refresh the cache immediately regardless of the <i>timeout</i> setting.

Example

The following example represents a cached JSP page:

```
<%@ taglib prefix="mypfx" uri="Sun Java System Application Server Tags" %>

<%
    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.

Table 3-2 flush Attributes

Attribute	Default	Description
key	<i>ServletPath_Suffix</i>	(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.

Examples

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

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

To flush the entire cache:

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

Compiling JSPs: The Command-Line Compiler

Sun Java System Application Server provides the following ways of compiling JSP 1.2 compliant source files into servlets:

- JSPs are automatically compiled at runtime.
- The `asadmin deploy` command has a `precompilejsp` option; see the *Sun Java System Application Server Developer's Guide*.
- The `sun-appserv-jspc` Ant task allows you to precompile JSPs; see the *Sun Java System Application Server Developer's Guide*.
- The `jspc` command line tool, described in this section, allows you to precompile JSPs at the command line.

To allow the JSP container to pick up the precompiled JSPs from a JAR file, you must disable dynamic reloading of JSPs. To do this, set the `reload-interval` property to `-1` in the `jsp-config` element of the `sun-web.xml` file. See [“JSP Elements” on page 130](#).

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.

Table 3-3 File Specifiers for the `jspc` Command

File Specifier	Description
<i>files</i>	One or more JSP files to be compiled.
<code>-webapp dir</code>	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.

Table 3-4 Basic `jspc` Options

Option	Description
<code>-q</code>	Enables quiet mode (same as <code>-v0</code>). Only fatal error messages are displayed.
<code>-d dir</code>	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 <code>jspc</code> is invoked.
<code>-p name</code>	Specifies the name of the target package for all specified JSPs, overriding the default package generation performed by the <code>-d</code> option.
<code>-c name</code>	Specifies the target class name of the first JSP compiled. Subsequent JSPs are unaffected.
<code>-uribase dir</code>	Specifies the URI directory to which compilations are relative. Applies only to JSP files listed in the command, and not to JSP files specified with <code>-webapp</code> . This is the location of each JSP file relative to the <code>uriroot</code> . If this cannot be determined, the default is <code>/</code> .
<code>-uriroot dir</code>	Specifies the root directory against which URI files are resolved. Applies only to JSP files listed in the command, and not to JSP files specified with <code>-webapp</code> . If this option is not specified, all parent directories of the first JSP page are searched for a <code>WEB-INF</code> subdirectory. The closest directory to the JSP page that has one is used. If none of the JSP's parent directories have a <code>WEB-INF</code> subdirectory, the directory from which <code>jspc</code> is invoked is used.
<code>-genclass</code>	Compiles the generated servlets into class files.

The following table shows the advanced *options* for the `jspc` command.

Table 3-5 Advanced `jspc` Options

Option	Description
<code>-v[level]</code>	Enables verbose mode. The <i>level</i> is optional; the default is 2. Possible <i>level</i> values are: <ul style="list-style-type: none"> • 0 - fatal error messages only • 1 - error messages only • 2 - error and warning messages only • 3 - error, warning, and informational messages • 4 - error, warning, informational, and debugging messages
<code>-mapped</code>	Generates separate <code>write</code> calls for each HTML line and comments that describe the location of each line in the JSP file. By default, all adjacent <code>write</code> calls are combined and no location comments are generated.
<code>-die[code]</code>	Returns the error number specified by <i>code</i> if an error occurs. If the <i>code</i> is absent or unparseable it defaults to 1.
<code>-webinc file</code>	Creates partial servlet mappings for the <code>-webapp</code> option, which can be pasted into a <code>web.xml</code> file.
<code>-webxml file</code>	Creates an entire <code>web.xml</code> file for the <code>-webapp</code> option.
<code>-ieplugin class_id</code>	Specifies the Java plug-in COM class ID for Internet Explorer. Used by the <code><jsp:plugin></code> tags.

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 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.

Debugging JSPs

When you use Sun ONE Studio 5 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 ONE Studio, see the *Sun Java System Application Server Developer's Guide*. For further details, see the *Sun ONE Studio 5, Enterprise Edition Tutorial*.

Creating and Managing User Sessions

This chapter describes how to create and manage a session that allows users and transaction information to persist between interactions.

This chapter contains the following sections:

- [Introducing Sessions](#)
- [How to Use Sessions](#)
- [Enabling the Availability Service \(Enterprise Edition\)](#)
- [Session Managers](#)

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 (like handles to EJB components or database result sets) and authenticated user identities, among many interactions. For example, a session could 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 Application Server supports the servlet standard session interface, called `HttpSession`, for all session activities. This interface enables you to write portable, secure servlets.

The rest of this section includes these topics:

- [Sessions and Cookies](#)
- [Sessions and URL Rewriting](#)
- [Sessions and Security](#)
- [Distributed Sessions and Persistence \(Enterprise Edition\)](#)

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 browser 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 in the following ways:

- You can configure the [session-properties](#) for the entire web container:
 - To use the Administration interface, select Containers and then Web Container under your server instance. Then click on the Properties button under Session Configuration Properties. After editing the properties, click on OK, then Save.
 - You can also edit the [session-properties](#) element in the `server.xml` file, which is the same as the corresponding element in the `sun-web.xml` file. For details, see the *Sun Java System Application Server Administrator's Configuration File Reference*.
- You can configure the [session-properties](#) and [cookie-properties](#) elements in the `sun-web.xml` file for an individual web application. The `sun-web.xml` file is described in [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Sessions and URL Rewriting

You can configure whether sessions use URL rewriting in the following ways:

- You can configure the [session-properties](#) for the entire web container:

- To use the Administration interface, select Containers and then Web Container under your server instance. Then click on the Properties button under Session Configuration Properties. After editing the properties, click on OK, then Save.
 - You can also edit the `session-properties` element in the `server.xml` file, which is the same as the corresponding element in the `sun-web.xml` file. For details, see the *Sun Java System Application Server Administrator's Configuration File Reference*.
- You can configure the `session-properties` element in the `sun-web.xml` file for an individual web application. The `sun-web.xml` file is described in [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Sessions and Security

The Sun Java System Application Server 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 in to the session. Each interaction step from the servlet that receives an EJB request does two things: generates content for a JSP to format the output, and checks that the user is properly authenticated.

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 5, “Securing Web Applications.”](#)

Distributed Sessions and Persistence (Enterprise Edition)

A distributed session can run in multiple Sun Java System Application Server instances, provided the following criteria are met:

- Each server instance has the same distributable web application deployed to it. The `web-app` element of the `web.xml` deployment descriptor file must have the `distributable` subelement specified.
- Each server instance has access to the same Sun Java System high-availability database (HADB). For information about how to enable this database, see [“Enabling the Availability Service \(Enterprise Edition\)” on page 74](#).
- The web application uses high-availability session persistence. If a non-distributable web application is configured to use high-availability session persistence, an error is written to the server log. See [“PersistentManager” on page 80](#).

- All objects bound into a distributed session must be of the types listed in the “[Object Types Supported for J2EE Web Application Session State Failover](#)” table.

NOTE Contrary to the Servlet 2.3 specification, Sun Java System Application Server does not throw an `IllegalArgumentException` if an object type not supported for failover is bound into a distributed session.

You should keep the distributed session size as small as possible. Session size has a direct impact on overall system throughput.

A servlet that is not deployed as part of a web application is implicitly deployed to a default web application and has the default `ServletContext`. The default `ServletContext` is not distributed. (A web application with an empty context root does *not* have the default `ServletContext`.)

In the event of an instance or hardware failure, another server instance can take over a distributed session, with the following limitations:

- If a distributable web application references a J2EE component or resource, the reference may be lost. See the “[Object Types Supported for J2EE Web Application Session State Failover](#)” table for a list of the types of references that `HTTPSession` failover supports.
- References to open files or network connections are lost.

For information about how to work around these limitations, see the *Sun Java System Application Server Application Design Guidelines for Storing Session State*.

In the following table, *No* indicates that failover for the object type may not work in all cases and that no failover support is provided. However, failover may work in some cases for that object type. For example, failover may work because the class implementing that type is serializable.

For more information about the `InitialContext`, transaction recovery, and Administered Objects, see the *Sun Java System Application Server Developer's Guide to J2EE Services and APIs*. For more information about local and remote EJB references, see the *Sun Java System Application Server Developer's Guide to Enterprise JavaBeans Technology*.

Table 4-1 Object Types Supported for J2EE Web Application Session State Failover

Java Object Type	Failover Support
EntityBean local home reference, local object reference	Yes
Stateful SessionBean local home reference	Yes

Table 4-1 Object Types Supported for J2EE Web Application Session State Failover (*Continued*)

Java Object Type	Failover Support
Stateful SessionBean local object reference	Yes
Stateless SessionBean local home reference, local object reference	Yes
Co-located EntityBean remote home reference, remote reference	Yes
Co-located Stateful SessionBean remote home reference	Yes
Co-located Stateful SessionBean remote reference	Yes
Co-located Stateless SessionBean remote home reference, remote reference	Yes
Distributed EntityBean remote home reference, remote reference	Yes
Distributed Stateful SessionBean remote home reference, remote reference	Yes
Distributed Stateless SessionBean remote home reference, remote reference	Yes
JNDI Context	Yes, InitialContext and <code>java:comp/env</code>
UserTransaction	Yes, but if the instance that fails is never restarted, any prepared global transactions are lost and may not be correctly rolled back or committed
JDBC DataSource	No
Java™ Message Service (JMS) ConnectionFactory, Destination	No
JavaMail™ Session	No
Connection Factory	No
Administered Object	No
Web service reference	No
Serializable Java types	Yes

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.

The rest of this section includes these topics:

- [Creating or Accessing a Session](#)
- [Examining Session Properties](#)
- [Binding Data to a Session](#)
- [Invalidating a Session](#)

Creating or Accessing a Session

To create a new session or to 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 which is encapsulated in the request object. Calling the method with no arguments, creates a session if one does not already exist which 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 which 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. Otherwise the `SetCookie` string is placed in the HTTP response body instead of the HTTP header.

For more information about `getSession()`, see the Java Servlet Specification v2.3.

Examining Session Properties

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

The following table shows the methods to examine session properties.

Table 4-2 `HttpSession` Methods

<code>HttpSession</code> method	Description
<code>getCreationTime()</code>	Returns the session time in milliseconds since January 1, 1970, 00:00:00 GMT.
<code>getId()</code>	Returns the assigned session identifier. An HTTP session's identifier is a unique string which is created and maintained by the server.
<code>getLastAccessedTime()</code>	Returns the last time the client sent a request carrying the assigned session identifier (or -1 if its a new session) in milliseconds since January 1, 1970, 00:00:00 GMT.
<code>isNew()</code>	Returns a Boolean value indicating if the session is new. Its 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 <i>acknowledged</i> or <i>joined</i> 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 to examine servlet request properties.

Table 4-3 `HttpServletRequest` Methods

<code>HttpServletRequest</code> Methods	Description
<code>getRemoteUser()</code>	Gets the requesting user name (HTTP authentication can provide the information). Returns null if the request has no user name information.
<code>getRequestedSessionId()</code>	Returns the session ID specified with the request. This 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.
<code>isRequestedSessionIdValid()</code>	Checks if the request is associated to a currently valid session. If the session requested is not valid, it is not returned through the <code>getSession()</code> method.
<code>isRequestedSessionIdFromCookie()</code>	Returns true if the request's session ID provided by the client is a cookie, or false otherwise.
<code>isRequestedSessionIdFromURL()</code>	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 in order 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.

Table 4-4 HttpSession Methods

HttpSession Methods	Description
<code>getAttribute()</code>	Returns the object bound to a given name in the session or null if there is no such binding.
<code>getAttributeNames()</code>	Returns an array of names of all attributes bound to the session.
<code>setAttribute()</code>	Binds the specified object into the session with the given name. Any existing binding with the same name is overwritten. An object bound into a distributed session must implement the <code>java.io.Serializable</code> interface. See “Distributed Sessions and Persistence (Enterprise Edition)” on page 67 .
<code>removeAttribute()</code>	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 Application Server notifies the object under consideration, through the `HttpSessionBindingListener` interface, that it is being bound into or unbound from the session.

Invalidating a Session

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

TIP The session API does not provide an explicit session logout API, so any *logout* implementation must call the `session.invalidate()` API.

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 `sun-web.xml` deployment descriptor file. For more information, see the [session-properties](#) element in [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Enabling the Availability Service (Enterprise Edition)

This section describes how to enable the Sun Java System high-availability database (HADB) for session persistence. For information about how to set up and configure this database, see the *Sun Java System Application Server Administrator's Guide*. You should set the same availability value for all instances in a cluster to ensure consistent behavior.

Availability can be enabled at five different levels:

1. The server instance
2. The web container
3. The application
4. The web module
5. The web application itself

For availability to be enabled at a given level, it must be enabled at all higher levels as well. For example, to enable availability at the application level, you must also enable it at the server instance and web container levels.

The default for a given level is the setting at the next level up. For example, if availability is enabled at the web container level, it is enabled by default at the application level.

When availability is disabled at the server instance level (the default setting), enabling it at any other level has no effect. When availability is enabled at the server instance level, it is enabled at all levels unless explicitly disabled.

The following sections describe how to enable availability:

- [Server Instance and Web Container Levels](#)
- [Application and Web Module Levels](#)
- [Web Application Level](#)

Server Instance and Web Container Levels

You can enable availability at the server instance and web container level in these ways:

- [Using the Administration Interface](#)
- [Using the asadmin Command](#)
- [Editing the server.xml File](#)

Using the Administration Interface

To enable availability at the instance and web container level using the Administration interface, follow these steps:

1. Open the Availability Service component under your server instance.
2. Go to the Availability Service page.
3. Check the Instance Level Availability box.
4. Make sure that Web Container Availability is set to either Enabled or Specified by Instance.
5. Click on the Save button.
6. Click Properties under Persistence Store Properties.
7. In the Name field, type `store-pool-jndi-name`.
8. In the Value field, type the JNDI name of the HADB JDBC Resource. The assumed default is `jdbc/hastore`. For more information, see the *Administration Guide*.
9. Click on the Save button.
10. Go to the server instance page.
11. Apply Changes and restart the server.

Using the asadmin Command

To enable availability at the web container level, use the `asadmin set` command as follows, then restart the server:

```
asadmin set --user admin_user [--password admin_password] [--passwordfile
password_file] [--host localhost] [--port 4848] [--secure | -s]
instance_name.web-container.availabilityEnabled=true
```

Editing the server.xml File

The `store-pool-jndi-name` property in the `server.xml` file specifies that the HADB is used for session persistence. Note that `availability-enabled="true"` must also be set. The element hierarchy in the `server.xml` file looks like this when the HADB is configured and availability is enabled for the server instance:

```
<server name="server1" ... >
...
<availability-service availability-enabled="true">
  <persistence-store>
    <property name="store-pool-jndi-name" value="jdbc/hastore" />
  </persistence-store>
</availability-service>
...
</server>
```

For information about how to configure the persistence store, see the *Administration Guide*.

To enable availability at the web container level, set `availability-enabled="true"` in the `web-container` element of the `server.xml` file as follows, then restart the server:

```
<server name="server1">
...
  <web-container ... availability-enabled="true"/>
...
</server>
```

Application and Web Module Levels

You can enable availability at the application or web module level during deployment. For details, see [“Deploying Web Applications” on page 96](#).

As an alternative, you can edit the `server.xml` file. To enable availability at the application level, set `availability-enabled="true"` in the `j2ee-application` element of the `server.xml` file as follows, then restart the server:

```
<server name="server1">
...
  <applications>
    ...
    <j2ee-application
      name="MyApp"
      location="instance_dir/applications/j2ee-apps/MyApp"
      availability-enabled="true">
    </j2ee-application>
```

```

    ...
  </applications>
  ...
</server>

```

To enable availability at the web module level, set `availability-enabled="true"` in the `web-module` element of the `server.xml` file as follows, then restart the server:

```

<server name="server1">
  ...
  <applications>
    ...
    <web-module
      name="MyWebApp"
      context-root=""
      location="instance_dir/applications/j2ee-modules/MyWebApp"
      availability-enabled="true">
    </web-module>
    ...
  </applications>
  ...
</server>

```

Web Application Level

If the Availability Service is disabled, there is no high availability for HTTP session persistence. In other words, `persistence-type=memory`.

If the Availability Service is enabled, high availability for HTTP session persistence is enabled for all applications by default. If no further configuration exists either in the `server.xml` or `sun-web.xml` file, the default session persistence configuration is as follows:

```

persistence-type=ha
persistenceFrequency=time-based
persistenceScope=session

```

The availability setting applies unconditionally to all web applications deployed to the server instance with the `persistence-type` set to `ha`.

You can override this default configuration for all applications in `server.xml`. For details, see the *Sun Java System Application Server Administrator's Configuration File Reference*. You can override this default configuration for a specific application in `sun-web.xml`. For details, see [“Session Managers” on page 78](#).

The Availability Service setting also determines whether single sign-on, if configured, is highly available.

NOTE Enabling the Availability Service is not sufficient for configuring session persistence. For details, see [“Distributed Sessions and Persistence \(Enterprise Edition\)” on page 67](#).

Session Managers

A session manager automatically creates new session objects whenever a new session starts. In some circumstances, clients do not join the session, for example, if the session manager uses cookies and the client does not accept cookies. Sun Java System Application Server 7 gives you these session management options:

- [StandardManager](#), the default session manager
- [PersistentManager](#), a provided session manager that uses a persistent data store

NOTE The session manager interface is Unstable. An unstable interface may be experimental or transitional, and hence may change incompatibly, be removed, or be replaced by a more stable interface in the next release.

If the session manager configuration contains an error, the error is written to the server log and the `StandardManager` default configuration is used.

StandardManager

The `StandardManager` is the default session manager. By default this session manager provides no session persistence. However, you can configure it so that the session state is written to the file system prior to server shutdown. This session manager is not designed for a production environment.

Enabling StandardManager

To specify `StandardManager` for the entire web container, perform these tasks:

1. Open the Containers component under your server instance.
2. Go to the Web Container page.
3. Click on the Session Manager tab.

4. Change the Persistence Type to Memory.
5. Click on the Properties button under Properties.
6. Enter the names and values of any properties that you want to change from the default values. For a list of properties, see [“Manager Properties for StandardManager” on page 79](#).
7. Click on the OK button.
8. Click on the Save button.

To specify `StandardManager` for a specific web application, edit the `sun-web.xml` file as in the following example. The `persistence-type` property is optional. This overrides the web container settings for the web application.

```
<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 6, “Assembling and Deploying Web Modules.”](#)

Manager Properties for StandardManager

The following table describes `manager-properties` properties for the `StandardManager` session manager.

Table 4-5 `manager-properties` Properties

Property Name	Default Value	Description
<code>reapIntervalSeconds</code>	60	<p>Specifies the number of seconds between checks for expired sessions.</p> <p>You should set this value lower than the frequency at which session data changes. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed website, or you could lose the last few hits each time you restart the server.</p>

Table 4-5 `manager-properties` Properties (*Continued*)

Property Name	Default Value	Description
<code>maxSessions</code>	<code>-1</code>	Specifies the maximum number of sessions that can be in cache, or <code>-1</code> for no limit. After this, an attempt to create a new session causes an <code>IllegalStateException</code> to be thrown.
<code>sessionFilename</code>	<code>none</code> ; state is not preserved across restarts	Specifies the absolute or relative path to the directory in which the session state is preserved between application restarts, if preserving the state is possible. A relative path is relative to the temporary directory for this web application.

PersistentManager

The `PersistentManager` is the other session manager provided with Sun Java System Application Server. For session persistence, `PersistentManager` can use either of the following stores, to which each session is serialized:

- A file - Provides session persistence to the local file system, and allows a single server instance to recover the session state after a failure and restart. The session state is persisted in the background, and the rate at which this occurs is configurable. This store also provides passivation and activation of the session state to help control the amount of memory used. This option is not supported in a production environment.
- The Sun Java System high-availability database (HADB). The HADB allows sessions to be distributed. For details, see [“Distributed Sessions and Persistence \(Enterprise Edition\)” on page 67](#). In addition, you can configure the frequency and scope of session persistence. The HADB is also used as the passivation and activation store. Use this option in a production environment that requires session persistence.

You must enable the HADB before you can use distributed sessions. See [“Enabling the Availability Service \(Enterprise Edition\)” on page 74](#).

NOTE	The HADB setting is in the Sun Java System Application Server Enterprise Edition only.
-------------	--

Enabling PersistentManager

To specify `PersistentManager` for the entire web container, perform these tasks:

1. Open the Containers component under your server instance.
2. Go to the Web Container page.

3. Click on the Session Manager tab.
4. Change the Persistence Type to File or to Highly Available Database.
5. Click on the Properties button under Properties.
6. Enter the names and values of any properties that you want to change from the default values. For a list of properties, see [“Manager Properties for PersistentManager” on page 82.](#)
7. Click on the OK button.
8. Click on the Properties button under Session Store Properties.
9. Enter the names and values of any properties that you want to change from the default values. For a list of properties, see [“Store Properties for PersistentManager” on page 83.](#)
10. Click on the OK button.
11. Click on the Save button.

To specify `PersistentManager` for a specific web application, edit the `sun-web.xml` file as in the following example. Note that `persistence-type` must be set to `file` or `ha`. This overrides the web container settings for the web application.

```
<sun-web-app>
...
<session-config>
  <session-manager persistence-type=ha>
    <manager-properties>
      <property name=persistenceFrequency value=web-method />
    </manager-properties>
    <store-properties>
      <property name=persistenceScope value=session />
    </store-properties>
  </session-manager>
...
</session-config>
...
</sun-web-app>
```

For more information about the `sun-web.xml` file, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Manager Properties for PersistentManager

The following table describes `manager-properties` properties for the `PersistentManager` session manager.

Table 4-6 `manager-properties` Properties

Property Name	Default Value	Description
<code>reapIntervalSeconds</code>	60	<p>Specifies the number of seconds between checks for expired sessions.</p> <p>If the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>file</code> or <code>ha</code>, sessions are passivated if <code>maxSessions</code> has been exceeded.</p> <p>If <code>persistenceFrequency</code> is set to <code>time-based</code>, active sessions are stored at this interval.</p> <p>You should set this value lower than the frequency at which session data changes. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed website, or you could lose the last few hits each time you restart the server.</p>
<code>maxSessions</code>	-1	<p>Specifies the maximum number of sessions that can be in cache, or -1 for no limit. After this, an attempt to create a new session causes an <code>IllegalStateException</code> to be thrown.</p> <p>If the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>file</code> or <code>ha</code>, the session manager passivates sessions to the persistent store when this maximum is reached.</p>
<code>persistenceFrequency</code>	<code>web-method</code>	<p>Specifies how often the session state is stored. Allowed values are as follows:</p> <ul style="list-style-type: none"><code>web-method</code> - The session state is stored at the end of each web request prior to sending a response back to the client. This mode provides the best guarantee that the session state is fully updated in case of failure.<code>time-based</code> - The session state is stored in the background at the frequency set by <code>reapIntervalSeconds</code>. This mode provides less of a guarantee that the session state is fully updated. However, it can provide a significant performance improvement because the state is not stored after each request. <p>Applicable only if the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>ha</code>.</p>

Store Properties for PersistentManager

The following table describes `store-properties` properties for the `PersistentManager` session manager.

Table 4-7 `store-properties` Properties

Property Name	Default Value	Description
directory	<i>instance_dir/generated/jsp/j2ee-apps/appname/appname_war</i>	Specifies the absolute or relative path to the directory into which individual session files are written. A relative path is relative to the temporary work directory for this web application. Applicable only if the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>file</code> .
persistenceScope	session	Specifies how much of the session state is stored. Allowed values are as follows: <ul style="list-style-type: none"> <code>session</code> - The entire session state is stored every time. This mode provides the best guarantee that your session data is correctly stored for any distributable web application. <code>modified-session</code> - The entire session state is stored if it has been modified. A session is considered to have been modified if <code>HttpSession.setAttribute()</code> or <code>HttpSession.removeAttribute()</code> was called. You must guarantee that <code>setAttribute()</code> is called every time an attribute is changed. This is not a J2EE specification requirement, but it is required for this mode to work properly. <code>modified-attribute</code> - Only modified session attributes are stored. For this mode to work properly, you must follow some guidelines, which are explained immediately following this table. Applicable only if the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>ha</code> .

If the `persistenceScope` store property is set to `modified-attribute`, your web application should follow these guidelines:

- Call `setAttribute()` every time you modify the session state.

- Make sure there are no cross-references between attributes. The object graph under each distinct attribute key is serialized and stored separately. If there are any object cross references between the objects under each separate key, they are not serialized and deserialized correctly.
- Ideally, the session state should be stored in multiple attributes, or at least in a read-only attribute and a modifiable attribute.

NOTE The session persistence scope `modified-attribute` is not certified as a full production quality feature. You should evaluate the performance and stability of the web container under expected peak load with this persistence scope. If exceptions are logged or response time is too high, do not use this persistence scope for your production environment. Use `session` or `modified-session` instead.

Securing Web Applications

This chapter describes how to write a secure web application for the Sun Java System Application Server with components that perform user authentication and access authorization.

This chapter contains the following sections:

- [User Authentication by Servlets](#)
- [User Authentication for Single Sign-on](#)
- [User Authorization by Servlets](#)
- [Fetching the Client Certificate](#)
- [Security for SHTML and CGI](#)

User Authentication by Servlets

The web-based login mechanisms required by the J2EE Specification, v1.3 are supported by the Sun Java System Application Server. These mechanisms include:

- [HTTP Basic Authentication](#)
- [SSL Mutual Authentication](#)
- [Form-Based Login](#)

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 officially optional, but if it is not included, the server defaults to HTTP Basic Authentication, which is not very secure.

For more information regarding `web.xml` elements, see Chapter 13, “Deployment Descriptor,” of the Java Servlet Specification, v2.3.

For more information regarding `sun-web.xml` elements, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

For information about programmatic login, see the *Sun Java System Application Server Developer's Guide*.

HTTP Basic Authentication

HTTP basic authentication (RFC2068) is supported by the Sun Java System Application Server. Because passwords are sent with base64 encoding, this authentication type is not very secure. Use of SSL or another equivalent transport encryption protects the password during transmission.

SSL Mutual Authentication

Secure Socket Layer (SSL) 3.0 and the means to perform mutual (client/server) certificate-based authentication is a J2EE Specification, v1.3 requirement. This security mechanism provides user authentication using HTTPS (HTTP over SSL).

The Sun Java System Application Server SSL mutual authentication mechanism (also known as HTTPS authentication) supports the following cipher suites:

```
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
```

Form-Based Login

The login screen's look and feel cannot be controlled with the HTTP browser's built-in mechanisms. J2EE introduces 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 (an HTTP realm) and is used to authenticate previously unauthenticated users.

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

In order for the authentication to proceed appropriately, the login form action must always be `j_security_check`.

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 133](#).

User Authentication for Single Sign-on

The single sign-on across applications on the Sun Java System Application Server is supported by the Sun Java System Application Server 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 Application Server Administrator's Guide* or the *Sun Java System Application Server Administrator's Configuration File Reference*.

- 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 himself 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 to authenticate himself or herself again.

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 `virtual-server` element of the `server.xml` file:

- `sso-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 `true`.
- `sso-max-inactive-seconds` - 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.
- `sso-reap-interval-seconds` - Specifies the interval between purges of expired single sign-on records. The default value is 60.

Here is an example configuration with all default values:

```
<virtual-server id="server1" ... >
  ...
  <property name="sso-enabled" value="true"/>
  <property name="sso-max-inactive-seconds" value="300"/>
  <property name="sso-reap-interval-seconds" value="60"/>
</virtual-server>
```


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](#)
- [Defining Servlet Authorization Constraints](#)

Defining Roles

You define roles in the J2EE deployment descriptor file, `web.xml`, and the corresponding role mappings in the Sun Java System Application Server deployment descriptor file, `sun-application.xml` (or `sun-web.xml` for individually deployed web modules). For more information about `sun-web.xml`, see [Chapter 6, “Assembling and Deploying Web Modules.”](#)

Each [security-role-mapping](#) element in the `sun-application.xml` or `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 an individually deployed web module 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.

Note that for J2EE applications (EAR files), all security role mappings for the application modules must be specified in the `sun-application.xml` file. For individually deployed web modules, 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 default realm.

Defining Servlet Authorization Constraints

On the servlet level, you 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 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 Distinguished Name for the user.
            java.security.Principal userDN = clientCert.getSubjectDN();
            ...
        }
    }
}
```

The `userDn` is the fully qualified Distinguished Name for the user.

Security for SHTML and CGI

For security, server-parsed HTML tags and CGI scripts depend on the server's security configuration. The following J2EE-only security features are not available for server-parsed HTML tags and CGI scripts:

- J2EE realms
- J2EE roles
- Form-based login
- Single sign-on
- Programmatic login

- J2EE authorization constraints

For more information about the server's security configuration, see the *Sun Java System Application Server Administrator's Guide to Security*.

Assembling and Deploying Web Modules

This chapter describes how web modules are assembled and deployed in Sun Java System Application Server. For general assembly and deployment information, see the *Sun Java System Application Server Developer's Guide*.

The following topics are presented in this chapter:

- [Web Application Structure](#)
- [Creating Web Deployment Descriptors](#)
- [Deploying Web Applications](#)
- [Dynamic Reloading of Web Applications](#)
- [The sun-web-app_2_3-0.dtd File](#)
- [Elements in the sun-web.xml File](#)
- [Sample Web Module XML Files](#)

Web Application Structure

Web Applications have a directory structure, all 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 (web application 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/lib/*.jar`, the directory for JAR files containing beans and other utility 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 J2EE specification. Here is an example directory structure of a simple web application.

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

Here is an example directory structure of a simple J2EE application containing a web module.

```
+ converter_1/
|--- converterClient.jar
|--- META-INF/
|   |--- MANIFEST.MF
|   |--- application.xml
|   '--- sun-application.xml
|--- war-ic_war/
|   |--- index.jsp
|   |--- META-INF/
|   |   |--- MANIFEST.MF
|   '--- WEB-INF/
|       |--- web.xml
|       '--- sun-web.xml
|--- ejb-jar-ic_jar/
|   |--- Converter.class
|   |--- ConverterBean.class
|   |--- ConverterHome.class
|   '--- META-INF/
|       |--- MANIFEST.MF
|       |--- ejb-jar.xml
|       '--- sun-ejb-jar.xml
|--- app-client-ic_jar/
|   |--- ConverterClient.class
|   '--- META-INF/
|       |--- MANIFEST.MF
|       |--- application-client.xml
|       '--- sun-application-client.xml
```

Creating Web Deployment Descriptors

Sun Java System Application Server web modules include two deployment descriptor files:

- A J2EE standard file (`web.xml`), described in the Java Servlet Specification, v2.3, Chapter 13, “Deployment Descriptors.” You can find the specification here:
<http://java.sun.com/products/servlet/index.html>
- An optional Sun Java System Application Server specific file (`sun-web.xml`), described in this chapter.

The easiest way to create the `web.xml` and `sun-web.xml` files is to deploy a web module using the Administration interface or the Sun ONE Studio 5 IDE. For more information, see the next section or the *Sun Java System Application Server Developer's Guide*. For example `web.xml` and `sun-web.xml` files, see [“Sample Web Module XML Files” on page 134](#).

After you have created these files, you can edit them using the Sun ONE Studio 5 IDE or a combination of an editor and command line utilities such as Ant to reassemble and redeploy the updated deployment descriptor information. Apache Ant 1.4.1 is provided with Sun Java System Application Server. For more information, see the *Sun Java System Application Server Developer's Guide*.

Deploying Web Applications

When you deploy or undeploy a web application, you do not need to restart the server. When you redeploy a web application, you do not need to restart the server unless you are changing the deployment settings.

You can deploy a web application in these ways, which are described briefly:

- [Using the Command Line](#)
- [Using the Administration Interface](#)
- [Using Sun ONE Studio](#)
- [Deploying to a Cluster \(Enterprise Edition\)](#)

For more detailed information about deployment, see the *Sun Java System Application Server Developer's Guide*.

You can keep the generated source for JSPs by adding the `-keepgenerated` property to the `jsp-config` element in `sun-web.xml`. If you include this property when you deploy the web application, the generated source is kept in `instance_dir/generated/jsp/j2ee-apps/app_name/module_name` if it is in an application or `instance_dir/generated/jsp/j2ee-modules/module_name` if it is in an individually deployed web module.

NOTE	After a web application is undeployed, its <code>HttpSession</code> information is not immediately removed if sessions are persistent. <code>HttpSession</code> information is removed in the subsequent cycle, when timed out sessions are removed. Therefore, you should disable a web application before undeploying it if sessions are persistent.
-------------	--

Using the Command Line

To deploy a web application using the command line:

1. Edit the deployment descriptor files (`web.xml` and `sun-web.xml`) by hand.
2. Execute an Ant build command (such as `build war`) to reassemble the WAR module.
3. Write the web application to a WAR file if desired. This is optional. For example:

```
jar -cvf module_name.war *
```

4. Use the `asadmin deploy` command to deploy the WAR module. The syntax is as follows:

```
asadmin deploy --user admin_user [--password admin_password] [--passwordfile
password_file] --host hostname --port adminport [--secure | -s]
[--virtualservers virtual_servers] [--type application|ejb|web|connector]
[--contextroot contextroot] [--force=true] [--precompilejsp=false]
[--verify=false] [--name component_name] [--upload=true]
[--availabilityenabled] [--retrieve local_dirpath] [--instance instance_name]
filepath
```

NOTE The `--availabilityenabled` setting is in the Sun Java System Application Server Enterprise Edition only.

For example, the following command deploys a web application as an individual module:

```
asadmin deploy --user jadams --password secret --host localhost --port
4848 --type web --instance server1 myWebApp.war
```

If `upload` is set to `false`, the *filepath* must be an absolute path on the server machine.

Using the Administration Interface

To deploy a web application using the Administration interface:

1. Open the Applications component under your server instance.
2. Go to the Web Applications page.
3. Click on the Deploy button.
4. Enter the full path to the WAR module (or click on Browse to find it), then click on the OK button.

5. Enter the web application name and the context root.

You can also redeploy the web application if it already exists by checking the appropriate box. This is optional.

6. To enable availability, make sure Availability Enabled is set to either true or Specified by Container (if availability is enabled for the web container).
7. Assign the web application to one or more virtual servers by checking the boxes next to the virtual server names.
8. Click on the OK button.

NOTE	The Availability Enabled setting is in the Sun Java System Application Server Enterprise Edition only.
-------------	--

Using Sun ONE Studio

You can use Sun ONE Studio 5, to assemble and deploy web applications. For more information about using Sun ONE Studio, see the *Sun ONE Studio 5, Enterprise Edition Tutorial*.

NOTE	In Sun ONE Studio, deploying web application is referred to as <i>executing</i> it.
-------------	---

NOTE	Sun ONE Studio does not currently support the session persistence attributes related to high availability. To set these attributes, you must edit the <code>sun-web.xml</code> file by hand.
-------------	--

Deploying to a Cluster (Enterprise Edition)

If a server instance is part of a cluster, you should deploy the same modules and applications to each instance in the cluster using the `cladmin` command. For details, see the *Sun Java System Application Server Administrator's Guide*.

If you are setting up load balancing, web module context roots must be unique within a cluster. See the *Sun Java System Application Server Administrator's Guide* for more information about load balancing.

Dynamic Reloading of Web Applications

If you make code changes to a web application and *dynamic reloading* is enabled, you do not need to redeploy the web application or restart the server. To enable dynamic reloading, you can do one of the following:

- Use the Administration interface:
 - a. Open the Applications component under your server instance.
 - b. Go to the Applications page.
 - c. Check the Reload Enabled box to enable dynamic reloading.
 - d. Enter a number of seconds in the Reload Poll Interval field to set the interval at which applications and modules are checked for code changes and dynamically reloaded.
 - e. Click on the Save button.
 - f. Go to the server instance page and select the Apply Changes button.

For details, see the *Sun Java System Application Server Administrator's Guide*.

- Edit the following attributes of the `server.xml` file's `applications` element, then restart the server:
 - `dynamic-reload-enabled="true"` enables dynamic reloading.
 - `dynamic-reload-poll-interval-in-seconds` sets the interval at which applications and modules are checked for code changes and dynamically reloaded.

For details about `server.xml`, see the *Sun Java System Application Server Administrator's Configuration File Reference*.

In addition, to load new servlet files, reload EJB related changes, or reload deployment descriptor changes, you must do the following:

1. Create an empty file named `.reload` at the root of the deployed application:


```
instance_dir/applications/j2ee-apps/app_name/.reload
```

 or individually deployed module:


```
instance_dir/applications/j2ee-modules/module_name/.reload
```
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`.

The sun-web-app_2_3-0.dtd File

The `sun-web-app_2_3-0.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_3-0.dtd` file is located in the `install_dir/lib/dtds` directory.

NOTE Do not edit the `sun-web-app_2_3-0.dtd` file; its contents change only with new versions of Sun Java System Application Server.

For general information about DTD files and XML, see the XML specification at:

<http://www.w3.org/TR/REC-xml>

Each element defined in a DTD file (which may be present in the corresponding XML file) can contain the following:

- [Subelements](#)
- [Data](#)
- [Attributes](#)

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.

Table 6-1 Requirement Rules and Subelement Suffixes

Subelement Suffix	Requirement Rule
<i>element*</i>	Can contain <i>zero or more</i> of this subelement.
<i>element?</i>	Can contain <i>zero or one</i> of this subelement.
<i>element+</i>	Must contain <i>one or more</i> of this subelement.
<i>element</i> (no suffix)	Must contain <i>only one</i> 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 cache    max-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 Application Server 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](#)
- [Security Elements](#)
- [Session Elements](#)
- [Reference Elements](#)
- [Caching Elements](#)
- [Classloader Elements](#)
- [JSP Elements](#)
- [Internationalization Elements](#)

NOTE Subelements must be defined in the order in which they are listed under each **Subelements** heading unless otherwise noted.

General Elements

General elements are as follows:

- `sun-web-app`
- `property`
- `description`

sun-web-app

Defines Sun Java System Application Server specific configuration for a web module. 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.

Table 6-2 sun-web-app Subelements

Element	Required	Description
<code>security-role-mapping</code>	zero or more	Maps roles to users or groups in the currently active realm.
<code>servlet</code>	zero or more	Specifies a principal name for a servlet, which is used for the <code>run-as</code> role defined in <code>web.xml</code> .
<code>session-config</code>	zero or one	Specifies session manager, session cookie, and other session-related information.
<code>resource-env-ref</code>	zero or more	Maps the absolute JNDI name to the <code>resource-env-ref</code> in the corresponding J2EE XML file.
<code>resource-ref</code>	zero or more	Maps the absolute JNDI name to the <code>resource-ref</code> in the corresponding J2EE XML file.
<code>ejb-ref</code>	zero or more	Maps the absolute JNDI name to the <code>ejb-ref</code> in the corresponding J2EE XML file.
<code>cache</code>	zero or one	Configures caching for web application components.
<code>class-loader</code>	zero or one	Specifies classloader configuration information.
<code>jsp-config</code>	zero or one	Specifies JSP configuration information.
<code>locale-charset-info</code>	zero or one	Specifies internationalization settings.
<code>property</code>	zero or more	Specifies a property, which has a name and a value.

Attributes

none

Properties

The following table describes properties for the sun-web-app element.

Table 6-3 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 <code>ServletContext.getContext()</code> method.
tempdir	<i>instance_dir/generated/j2ee-apps/app_name</i> or <i>instance_dir/generated/j2ee-modules/module_name</i>	Specifies a temporary directory for use by this web module. This value is used to construct the value of the <code>javax.servlet.context.tempdir</code> context attribute. Compiled JSPs are also placed in this directory.
singleThreadedServletPoolSize	5	Specifies the maximum number of servlet instances allocated for each <code>SingleThreadModel</code> servlet in the web application.
useResponseCTForHeaders	false	If true, header data is sent in the encoding specified in the method <code>Response.setContentType()</code> . Use this property to send non-ASCII characters in the headers.

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 Application Server
- Needed by a system or object that Sun Java System Application Server 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` element's `persistence-type` attribute. For details, see the description of the [session-manager](#) element.

Subelements

The following table describes subelements for the `property` element.

Table 6-4 `property` Subelements

Element	Required	Description
<code>description</code>	zero or one	Specifies an optional text description of a property.

Attributes

The following table describes attributes for the `property` element.

Table 6-5 `property` Attributes

Attribute	Default	Description
<code>name</code>	<code>none</code>	Specifies the name of the property.
<code>value</code>	<code>none</code>	Specifies the value of the property.

description

Contains data that specifies a text description of the containing element.

Subelements

`none`

Attributes

`none`

Security Elements

Security elements are as follows:

- `security-role-mapping`
- `servlet`
- `servlet-name`
- `role-name`
- `principal-name`

- `group-name`

security-role-mapping

Maps roles to users or groups in the currently active realm. See the *Sun Java System Application Server Developer's Guide* for how to define the currently active realm.

Subelements

The following table describes subelements for the `security-role-mapping` element.

Table 6-6 `security-role-mapping` Subelements

Element	Required	Description
<code>role-name</code>	only one	Contains the role name.
<code>principal-name</code>	requires at least one <code>principal-name</code> or <code>group-name</code>	Contains a principal (user) name in the current realm.
<code>group-name</code>	requires at least one <code>principal-name</code> or <code>group-name</code>	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.

Table 6-7 `servlet` Subelements

Element	Required	Description
<code>servlet-name</code>	only one	Contains the name of a servlet, which is matched to a <code>servlet-name</code> in <code>web.xml</code> .
<code>principal-name</code>	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`
- `session-manager`
- `manager-properties`
- `store-properties`
- `session-properties`
- `cookie-properties`

NOTE The session manager interface is Unstable. An unstable interface may be experimental or transitional, and hence may change incompatibly, be removed, or be replaced by a more stable interface in the next release.

NOTE Sun ONE Studio does not currently support the session persistence attributes related to high availability. To set these attributes, you must edit the `sun-web.xml` file by hand.

session-config

Specifies session configuration information. Overrides the web container settings for an individual web application.

Subelements

The following table describes subelements for the `session-config` element.

Table 6-8 session-config Subelements		
Element	Required	Description
<code>session-manager</code>	zero or one	Specifies session manager configuration information.
<code>session-properties</code>	zero or one	Specifies session properties.
<code>cookie-properties</code>	zero or one	Specifies session cookie properties.

Attributes

none

session-manager

Specifies session manager information.

Subelements

The following table describes subelements for the `session-manager` element.

Table 6-9 `session-manager` Subelements

Element	Required	Description
manager-properties	zero or one	Specifies session manager properties.
store-properties	zero or one	Specifies session persistence (storage) properties.

Attributes

The following table describes attributes for the `session-manager` element.

Table 6-10 `session-manager` Attributes

Attribute	Default Value	Description
<code>persistence-type</code>	<code>memory</code>	(optional) Specifies the session persistence mechanism. Allowed values are <code>memory</code> , <code>file</code> , and <code>ha</code> . For production environments that require session persistence, use <code>ha</code> . The <code>custom</code> value is not implemented and should not be used.

NOTE	Setting the <code>persistence-type</code> to <code>ha</code> is not sufficient for configuring session persistence. For details, see “Distributed Sessions and Persistence (Enterprise Edition)” on page 67.
-------------	--

manager-properties

Specifies session manager properties.

Subelements

The following table describes subelements for the `manager-properties` element.

Table 6-11 `manager-properties` Subelements

Element	Required	Description
<code>property</code>	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.

Table 6-12 `manager-properties` Properties

Property Name	Default Value	Description
<code>reapIntervalSeconds</code>	60	<p>Specifies the number of seconds between checks for expired sessions.</p> <p>If the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>file</code> or <code>ha</code>, sessions are passivated if <code>maxSessions</code> has been exceeded.</p> <p>If <code>persistenceFrequency</code> is set to <code>time-based</code>, active sessions are stored at this interval.</p> <p>You should set this value lower than the frequency at which session data changes. For example, this value should be as low as possible (1 second) for a hit counter servlet on a frequently accessed website, or you could lose the last few hits each time you restart the server.</p>
<code>maxSessions</code>	-1	<p>Specifies the maximum number of sessions that can be in cache, or -1 for no limit. After this, an attempt to create a new session causes an <code>IllegalStateException</code> to be thrown.</p> <p>If the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>file</code> or <code>ha</code>, the session manager passivates sessions to the persistent store when this maximum is reached.</p>

Table 6-12 `manager-properties` Properties (Continued)

Property Name	Default Value	Description
<code>sessionFilename</code>	none; state is not preserved across restarts	<p>Specifies the absolute or relative path to the directory in which the session state is preserved between application restarts, if preserving the state is possible. A relative path is relative to the temporary directory for this web application.</p> <p>Applicable only if the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>memory</code>.</p>
<code>persistenceFrequency</code>	<code>web-method</code>	<p>Specifies how often the session state is stored. Allowed values are as follows:</p> <ul style="list-style-type: none"><code>web-method</code> - The session state is stored at the end of each web request prior to sending a response back to the client. This mode provides the best guarantee that the session state is fully updated in case of failure.<code>time-based</code> - The session state is stored in the background at the frequency set by <code>reapIntervalSeconds</code>. This mode provides less of a guarantee that the session state is fully updated. However, it can provide a significant performance improvement because the state is not stored after each request. <p>Applicable only if the <code>persistence-type</code> attribute of the <code>session-manager</code> element is <code>ha</code>.</p>

store-properties

Specifies session persistence (storage) properties.

Subelements

The following table describes subelements for the `store-properties` element.

Table 6-13 `store-properties` Subelements

Element	Required	Description
<code>property</code>	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.

Table 6-14 store-properties Properties

Property Name	Default Value	Description
directory	<i>instance_dir/generated/jsp/j2ee-apps/appname/appname_war</i>	<p>Specifies the absolute or relative pathname of the directory into which individual session files are written. A relative path is relative to the temporary work directory for this web application.</p> <p>Applicable only if the persistence-type attribute of the session-manager element is file.</p>
persistenceScope	session	<p>Specifies how much of the session state is stored. Allowed values are as follows:</p> <ul style="list-style-type: none">• session - The entire session state is stored every time. This mode provides the best guarantee that your session data is correctly stored for any distributable web application.• modified-session - The entire session state is stored if it has been modified. A session is considered to have been modified if HttpSession.setAttribute() or HttpSession.removeAttribute() was called. You must guarantee that setAttribute() is called every time an attribute is changed. This is not a J2EE specification requirement, but it is required for this mode to work properly.• modified-attribute - Only modified session attributes are stored. For this mode to work properly, you must follow some guidelines, which are explained immediately following this table. <p>Applicable only if the persistence-type attribute of the session-manager element is ha.</p>

If the persistenceScope store property is set to modified-attribute, your web application must follow these guidelines:

- Call setAttribute() every time you modify the session state.
- Make sure there are no cross-references between attributes. The object graph under each distinct attribute key is serialized and stored separately. If there are any object cross references between the objects under each separate key, they are not serialized and deserialized correctly.
- Distribute the session state across multiple attributes, or at least between a read-only attribute and a modifiable attribute.

NOTE

The session persistence scope `modified-attribute` is not certified as a full production quality feature. You should evaluate the performance and stability of the web container under expected peak load with this persistence scope. If exceptions are logged or response time is too high, do not use this persistence scope for your production environment. Use `session` or `modified-session` instead.

session-properties

Specifies session properties.

Subelements

The following table describes subelements for the `session-properties` element.

Table 6-15 `session-properties` Subelements

Element	Required	Description
<code>property</code>	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.

Table 6-16 `session-properties` Properties

Property Name	Default Value	Description
<code>timeoutSeconds</code>	600	<p>Specifies the default maximum inactive interval (in seconds) for all sessions created in this web module. If set to 0 or less, sessions in this web module never expire.</p> <p>If a <code>session-timeout</code> element is specified in the <code>web.xml</code> file, the <code>session-timeout</code> value overrides any <code>timeoutSeconds</code> value. If neither <code>session-timeout</code> nor <code>timeoutSeconds</code> is specified, the <code>timeoutSeconds</code> default is used.</p> <p>Note that the <code>session-timeout</code> element in <code>web.xml</code> is specified in minutes, not seconds.</p>
<code>enableCookies</code>	true	Uses cookies for session tracking if set to true.

Table 6-16 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 <code>encodeURL</code> or <code>encodeRedirectURL</code> call in the servlet or JSP.
idLengthBytes	128	Specifies the number of bytes in this web module's session ID.

cookie-properties

Specifies session cookie properties.

Subelements

The following table describes subelements for the `cookie-properties` element.

Table 6-17 cookie-properties Subelements

Element	Required	Description
<code>property</code>	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.

Table 6-18 cookie-properties Properties

Property Name	Default Value	Description
cookiePath	Context path at which the web module is installed.	Specifies the pathname that is set when the cookie is created. The browser sends the cookie if the pathname for the request contains this pathname. If set to <code>/</code> (slash), the browser sends cookies to all URLs served by the Sun Java System Application Server. You can set the path to a narrower mapping to limit the request URLs to which the browser sends cookies.
cookieMaxAgeSeconds	-1	Specifies the expiration time (in seconds) after which the browser expires the cookie.
cookieDomain	(unset)	Specifies the domain for which the cookie is valid.

Table 6-18 cookie-properties Properties (*Continued*)

Property Name	Default Value	Description
cookieComment	Sun Java System Application 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`
- `resource-env-ref-name`
- `resource-ref`
- `res-ref-name`
- `default-resource-principal`
- `name`
- `password`
- `ejb-ref`
- `ejb-ref-name`
- `jndi-name`

NOTE	If a distributable web application references a component, and the web application's session fails over, the reference may be lost. For information about how to work around this limitation, see “Distributed Sessions and Persistence (Enterprise Edition)” on page 67 and the <i>Sun Java System Application Server Application Design Guidelines for Storing Session State</i> .
-------------	--

resource-env-ref

Maps the `res-ref-name` in the corresponding J2EE `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.

Table 6-19 resource-env-ref Subelements

Element	Required	Description
resource-env-ref-name	only one	Specifies the res-ref-name in the corresponding J2EE web.xml file resource-env-ref entry.
jndi-name	only one	Specifies the absolute jndi-name of a resource.

Attributes

none

resource-env-ref-name

Contains data that specifies the res-ref-name in the corresponding J2EE web.xml file resource-env-ref entry.

Subelements

none

Attributes

none

resource-ref

Maps the res-ref-name in the corresponding J2EE web.xml file resource-ref entry to the absolute jndi-name of a resource.

Subelements

The following table describes subelements for the resource-ref element.

Table 6-20 resource-ref Subelements

Element	Required	Description
res-ref-name	only one	Specifies the res-ref-name in the corresponding J2EE web.xml file resource-ref entry.
jndi-name	only one	Specifies the absolute jndi-name of a resource.
default-resource-principal	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 J2EE `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 the Sun Java™ System Message Queue broker user repository. See the “Security Management” chapter in the *Sun Java System Message Queue Administrator’s Guide* for details.

Subelements

The following table describes subelements for the `default-resource-principal` element.

Table 6-21 `default-resource-principal` Subelements

Element	Required	Description
<code>name</code>	only one	Contains the name of the principal.
<code>password</code>	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

ejb-ref

Maps the `ejb-ref-name` in the corresponding J2EE `ejb-jar.xml` file `ejb-ref` entry to the absolute `jndi-name` of a resource.

Subelements

The following table describes subelements for the `ejb-ref` element.

Table 6-22 `ejb-ref` Subelements

Element	Required	Description
<code>ejb-ref-name</code>	only one	Specifies the <code>ejb-ref-name</code> in the corresponding J2EE <code>ejb-jar.xml</code> file <code>ejb-ref</code> entry.
<code>jndi-name</code>	only one	Specifies the absolute <code>jndi-name</code> of a resource.

Attributes

none

ejb-ref-name

Contains data that specifies the `ejb-ref-name` in the corresponding J2EE `ejb-jar.xml` file `ejb-ref` entry.

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 Application Server 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 47](#). For details about JSP caching, see [“JSP Caching” on page 58](#).

Caching elements are as follows:

- `cache`
- `cache-helper`
- `default-helper`
- `cache-mapping`
- `url-pattern`
- `timeout`

- `http-method`
- `key-field`
- `constraint-field`
- `value`

cache

Configures caching for web application components.

Subelements

The following table describes subelements for the `cache` element.

Table 6-23 `cache` Subelements

Element	Required	Description
<code>cache-helper</code>	zero or more	Specifies a custom class that implements the <code>CacheHelper</code> interface.
<code>default-helper</code>	zero or one	Allows you to change the properties of the default, built-in <code>cache-helper</code> class.
<code>property</code>	zero or more	Specifies a cache property, which has a name and a value.
<code>cache-mapping</code>	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.

Table 6-24 `cache` Attributes

Attribute	Default Value	Description
<code>max-entries</code>	4096	(optional) Specifies the maximum number of entries the cache can contain. Must be a positive integer.
<code>timeout-in-seconds</code>	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 <code>timeout</code> element.

Table 6-24 cache Attributes (*Continued*)

Attribute	Default Value	Description
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.

Table 6-25 cache Properties

Property	Default Value	Description
cacheClassName	<code>com.sun.appserv.web.cache.LruCache</code>	Specifies the fully qualified name of the class that implements the cache functionality. The “ cacheClassName Values ” table below lists possible values.
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 <code>cacheClassName</code> is set to <code>com.sun.appserv.web.cache.MultiLruCache</code> .
MaxSize	unlimited; <code>Long.MAX_VALUE</code>	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 <code>cacheClassName</code> is set to <code>com.sun.appserv.web.cache.BoundedMultiLruCache</code> .

Cache Class Names

The following table lists possible values of the `cacheClassName` property.

Table 6-26 cacheClassName Values

Value	Description
<code>com.sun.appserv.web.cache.LruCache</code>	A bounded cache with an LRU (least recently used) cache replacement policy.
<code>com.sun.appserv.web.cache.BaseCache</code>	An unbounded cache suitable if the maximum number of entries is known.
<code>com.sun.appserv.web.cache.MultiLruCache</code>	A cache suitable for a large number of entries (>4096). Uses the <code>MultiLRUSegmentSize</code> property.

Table 6-26 cacheClassName Values (Continued)

Value	Description
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 50.

Subelements

The following table describes subelements for the cache-helper element.

Table 6-27 cache-helper Subelements

Element	Required	Description
property	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the cache-helper element.

Table 6-28 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.

Table 6-29 default-helper Subelements

Element	Required	Description
property	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.

Table 6-30 default-helper Properties

Property	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 52 .

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.

Table 6-31 cache-mapping Subelements

Element	Required	Description
servlet-name	requires one servlet-name or url-pattern	Contains the name of a servlet.
url-pattern	requires one servlet-name or url-pattern	Contains a servlet URL pattern for which caching is enabled.

Table 6-31 cache-mapping Subelements (*Continued*)

Element	Required	Description
cache-helper-ref	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.
timeout	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.
refresh-field	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	zero or more if cache-helper-ref is not used	Contains an HTTP method that is eligible for caching.
key-field	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	zero or more if cache-helper-ref is not used	Specifies a cacheability constraint for the given <code>url-pattern</code> or <code>servlet-name</code> .

Attributes

none

url-pattern

Contains data that specifies a servlet URL pattern for which caching is enabled. See the Servlet 2.3 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.

Table 6-32 `timeout` Attributes

Attribute	Default Value	Description
<code>name</code>	<code>none</code>	Specifies the timeout input parameter, whose value is interpreted in seconds. The field's type must be <code>java.lang.Long</code> or <code>java.lang.Integer</code> .
<code>scope</code>	<code>request.attribute</code>	(optional) Specifies the scope in which the input parameter can be present. Allowed values are <code>context.attribute</code> , <code>request.header</code> , <code>request.parameter</code> , <code>request.cookie</code> , <code>request.attribute</code> , and <code>session.attribute</code> .

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.

Table 6-33 `refresh-field` Attributes

Attribute	Default Value	Description
<code>name</code>	<code>none</code>	Specifies the input parameter name.
<code>scope</code>	<code>request.parameter</code>	(optional) Specifies the scope in which the input parameter can be present. Allowed values are <code>context.attribute</code> , <code>request.header</code> , <code>request.parameter</code> , <code>request.cookie</code> , <code>session.id</code> , and <code>session.attribute</code> .

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 2.3 specification, section SRV 4.4, for details on the Servlet Path.

Subelements

`none`

Attributes

The following table describes attributes for the `key-field` element.

Table 6-34 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 cacheability 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.

Table 6-35 constraint-field Subelements

Element	Required	Description
value	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.

Table 6-36 constraint-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, request.attribute, and session.attribute.
cache-on-match	true	(optional) If true, caches the response if matching succeeds. Overrides the same attribute in a value subelement.

Table 6-36 constraint-field Attributes (*Continued*)

Attribute	Default Value	Description
cache-on-match-failure	false	(optional) If true, caches the response if matching fails. Overrides the same attribute in a value subelement.

value

Contains data that specifies a value to be matched to the input parameter value. The matching is case sensitive. For example:

```
<value match-expr="in-range">1-60</value>
```

Subelements

none

Attributes

The following table describes attributes for the `value` element.

Table 6-37 value Attributes

Attribute	Default Value	Description
match-expr	equals	(optional) Specifies the type of comparison performed with the value. Allowed values are equals, not-equals, greater, lesser, and in-range. If match-expr is greater or lesser, the value must be a number. If match-expr is in-range, the value must be of the form <i>n1-n2</i> , where <i>n1</i> and <i>n2</i> are numbers.
cache-on-match	true	(optional) If true, caches the response if matching succeeds.
cache-on-match-failure	false	(optional) If true, caches the response if matching fails.

ClassLoader Elements

ClassLoader elements are as follows:

- [class-loader](#)

class-loader

Configures the classloader for the web module.

Subelements

none

Attributes

The following table describes attributes for the `class-loader` element.

Table 6-38 `class-loader` Attributes

Attribute	Default Value	Description
<code>extra-class-path</code>	<code>null</code>	(optional) Specifies additional classpath settings for this web module.
<code>delegate</code>	<code>false</code>	<p>(optional) If <code>true</code>, the web module follows the standard classloader delegation model and delegates to its parent classloader first before looking in the local classloader. If <code>false</code>, the web module follows the delegation model specified in the Servlet specification and looks in its classloader before looking in the parent classloader.</p> <p>For a web component of a web service, you must set this value to <code>true</code>.</p> <p>Legal values are <code>on</code>, <code>off</code>, <code>yes</code>, <code>no</code>, <code>1</code>, <code>0</code>, <code>true</code>, <code>false</code>.</p>

NOTE

If the `delegate` flag is set to its default value of `false`, the classloader delegation behavior complies with the Servlet 2.3 specification, section 9.7.2. If set to `true`, classes and resources residing in container-wide library JAR files are loaded in preference to classes and resources packaged within the WAR file, contrary to what this specification recommends.

Portable programs that use this flag should not be packaged with any classes or interfaces that are a part of the J2EE specification. The behavior of a program that includes such classes or interfaces in its WAR file is undefined.

JSP Elements

JSP elements are as follows:

- `jsp-config`

jsp-config

Specifies JSP configuration information.

Subelements

The following table describes subelements for the `jsp-config` element.

Table 6-39 `jsp-config` Subelements

Element	Required	Description
<code>property</code>	zero or more	Specifies a property.

Attributes

none

Properties

The following table describes properties for the `jsp-config` element.

Table 6-40 `jsp-config` Properties

Property Name	Default Value	Description
<code>ieClassId</code>	<code>clsid:8AD9C840-044E-11D1-B3E9-00805F499D93</code>	The Java plug-in COM class ID for Internet Explorer. Used by the <code><jsp:plugin></code> tags.
<code>javaCompilerPlugin</code>	The internal JDK compiler (<code>javac</code>)	<p>The fully qualified class name of the Java compiler plug-in to be used. Not needed for the default compiler.</p> <p>For example, to use the <code>jikes</code> compiler for JSP pages, set the <code>javaCompilerPlugin</code> property to <code>org.apache.jasper.compiler.JikesJavaCompiler</code>, then set the <code>javaCompilerPath</code> property to point to the <code>jikes</code> executable.</p>
<code>javaCompilerPath</code>	none	Specifies the path to the executable of an out-of-process Java compiler such as <code>jikes</code> . Ignored for the default compiler. Needed only if the <code>javaCompilerPlugin</code> property is specified.

Table 6-40 jsp-config Properties (*Continued*)

Property Name	Default Value	Description
javaEncoding	UTF8	<p>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 <code>javaEncoding</code> value.</p> <p>For encodings you can use, see:</p> <p>http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html</p>
classdebuginfo	false	Specifies whether the generated Java servlets should be compiled with the debug option set (<code>-g</code> for <code>javac</code>).
keepgenerated	true	If set to <code>true</code> , keeps the generated Java files. If <code>false</code> , deletes the Java files.
largefile	false	If set to <code>true</code> , static HTML is stored in a separate data file when a JSP is compiled. This is useful when a JSP is very large, because it minimizes the size of the generated servlet.
mappedfile	false	If set to <code>true</code> , generates separate <code>write</code> calls for each HTML line and comments that describe the location of each line in the JSP file. By default, all adjacent <code>write</code> calls are combined and no location comments are generated.
scratchdir	The default work directory for the web application	The working directory created for storing all the generated code.
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.

Internationalization Elements

Internationalization elements are as follows:

- `locale-charset-info`
- `locale-charset-map`
- `parameter-encoding`

locale-charset-info

Specifies information about the application's internationalization settings.

Subelements

The following table describes subelements for the `locale-charset-info` element.

Table 6-41 `locale-charset-info` Subelements

Element	Required	Description
<code>locale-charset-map</code>	one or more	Maps a locale and an agent to a character set.
<code>parameter-encoding</code>	zero or one	Determines how the web container decodes parameters from forms for this web application according to a hidden field value.

Attributes

The following table describes attributes for the `locale-charset-info` element.

Table 6-42 `locale-charset-info` Attributes

Attribute	Default Value	Description
<code>default-locale</code>	none	Specifies the default locale.

locale-charset-map

Maps locales and agents to character sets.

For encodings you can use, see:

<http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html>

Subelements

The following table describes subelements for the `locale-charset-map` element.

Table 6-43 `locale-charset-map` Subelements

Element	Required	Description
description	zero or one	Specifies an optional text description of a mapping.

Attributes

The following table describes attributes for the `locale-charset-map` element.

Table 6-44 `locale-charset-map` Attributes

Attribute	Default Value	Description
<code>locale</code>	<code>none</code>	Specifies the locale name.
<code>agent</code>	<code>none</code>	(optional) Specifies the type of client that interacts with the application server. For a given locale, different agents may have different preferred character sets. The value of this attribute must exactly match the value of the <code>user-agent</code> HTTP request header sent by the client. See the “Example agent Attribute Values” table for more information.
<code>charset</code>	<code>none</code>	Specifies the character set.

Example Agents

The following table specifies example agent attribute values.

Table 6-45 Example agent Attribute Values

Agent	user-agent Header and agent Attribute Value
Netscape 4.7 for Solaris	Mozilla/4.7 [en] (X11; u; Sun OS 5.6 sun4u)

parameter-encoding

Specifies a hidden field 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/docs/guide/intl/encoding.doc.html>

Subelements

none

Attributes

The following table describes attributes for the `parameter-encoding` element.

Table 6-46 `parameter-encoding` Attributes

Attribute	Default Value	Description
<code>form-hint-field</code>	<code>none</code>	The name of the hidden field in the form that specifies the parameter encoding.

Sample Web Module XML Files

This section includes the following:

- [Sample web.xml File](#)
- [Sample sun-web.xml File](#)

Sample web.xml File

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

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" 'http://java.sun.com/j2ee/dtds/web-app_2_2.dtd'>

<web-app>

    <display-name>webapps-simple</display-name>

    <description>

        The jakarta-tomcat-4.0.3 sample apps ports over to S1AS.

    </description>

    <distributable></distributable>

    <servlet>

        <servlet-name>HelloWorldExample</servlet-name>

        <servlet-class>

            samples.webapps.simple.servlet.HelloWorldExample
```

```

        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>RequestHeaderExample</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.RequestHeaderExample
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>SnoopServlet</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.SnoopServlet
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>servletToJsp</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.servletToJsp
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>RequestInfoExample</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.RequestInfoExample
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>SessionExample</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.SessionExample

```

```
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>CookieExample</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.CookieExample
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>RequestParamExample</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.RequestParamExample
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>SendMailServlet</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.SendMailServlet
        </servlet-class>
    </servlet>
    <servlet>
        <servlet-name>JndiServlet</servlet-name>
        <servlet-class>
            samples.webapps.simple.servlet.JndiServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldExample</servlet-name>
        <url-pattern>/helloworld</url-pattern>
    </servlet-mapping>
```



```

<servlet-mapping>
    <servlet-name>RequestHeaderExample</servlet-name>
    <url-pattern>/requestheader</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SnoopServlet</servlet-name>
    <url-pattern>/snoop</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>servletToJsp</servlet-name>
    <url-pattern>/servletToJsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>RequestInfoExample</servlet-name>
    <url-pattern>/requestinfo</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SessionExample</servlet-name>
    <url-pattern>/session</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>CookieExample</servlet-name>
    <url-pattern>/cookie</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>RequestParamExample</servlet-name>
    <url-pattern>/requestparam</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SendMailServlet</servlet-name>

```

```

        <url-pattern>/SendMailServlet</url-pattern>
    </servlet-mapping>
</servlet-mapping>

    <servlet-name>JndiServlet</servlet-name>

    <url-pattern>/JndiServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
<taglib>
    <taglib-uri>
        http://java.apache.org/tomcat/examples-taglib
    </taglib-uri>
    <taglib-location>
        /WEB-INF/tlds/example-taglib.tld
    </taglib-location>
</taglib>
<resource-ref>
    <res-ref-name>mail/Session</res-ref-name>
    <res-type>javax.mail.Session</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
<!-- Define the context-relative URL(s) to be protected -->
        <url-pattern>/jsp/security/protected/*</url-pattern>
<!-- If you list http methods, only those methods are protected -->
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>

```

```

        <http-method>POST</http-method>
        <http-method>PUT</http-method>
    </web-resource-collection>
    <auth-constraint>
<!-- Anyone with one of the listed roles may access this area -->
        <role-name>tomcat</role-name>
        <role-name>role1</role-name>
    </auth-constraint>
</security-constraint>
<!-- Environment entry examples -->
    <env-entry>
        <description>
            The maximum number of tax exemptions allowed to be set.
        </description>
        <env-entry-name>maxExemptions</env-entry-name>
        <env-entry-value>15</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>minExemptions</env-entry-name>
        <env-entry-value>1</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/name1</env-entry-name>
        <env-entry-value>value1</env-entry-value>
        <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/bar/name2</env-entry-name>

```

```

        <env-entry-value>true</env-entry-value>
        <env-entry-type>java.lang.Boolean</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>name3</env-entry-name>
        <env-entry-value>1</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/name4</env-entry-name>
        <env-entry-value>10</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
</web-app>

```

Sample sun-web.xml File

```

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

<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Sun Java
System Application Server 7.0 Servlet 2.3//EN"
'http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-0.dtd'>

<sun-web-app>
    <session-config>
        <session-manager/>
    </session-config>
    <resource-ref>
        <res-ref-name>mail/Session</res-ref-name>
        <jndi-name>mail/Session</jndi-name>
    </resource-ref>
    <jsp-config/>
</sun-web-app>

```

Using Server-Parsed HTML

HTML files can contain tags that are executed on the server. In addition to supporting the standard server-side tags, Sun Java System Application Server 7 allows you to embed servlets and define your own server-side tags.

You can create custom server-parsed HTML tags. For more information, see the *Sun Java System Application Server NSAPI Developer's Guide*.

For security, server-parsed HTML tags depend on the server's security configuration. For more information, see [“Security for SHTML and CGI” on page 90](#) and the *Sun Java System Application Server Administrator's Guide to Security*.

This chapter has the following sections:

- [Server-Side HTML and J2EE Web Applications](#)
- [Enabling Server-Side HTML](#)
- [Using Server-Side HTML Commands](#)
- [Embedding Servlets](#)
- [Time Formats](#)

Server-Side HTML and J2EE Web Applications

In Sun Java System Application Server, server-parsed HTML cannot interoperate with J2EE web applications. Specifically:

- Do not place server-parsed HTML within web application context roots.
- Do not include the output of server-parsed HTML in servlets or JSPs.
- Do not forward requests to server-parsed HTML from servlets or JSPs.

- You cannot apply J2EE security-constraint and filter-mapping features to server-parsed HTML.

Enabling Server-Side HTML

To enable server-side HTML:

1. Open the HTTP Server component under your server instance in the Administration interface.
2. Go to the Virtual Servers page.
3. Click on the name of the virtual server for which you are enabling server-side HTML.
4. Click on the HTTP/HTML tab.
5. Click on the Parse HTML option.
6. Choose a resource for which the server will parse HTML.

Choose the virtual server or a specific directory within the virtual server.

If you choose a directory, the server will parse HTML only when the server receives a URL for that directory or any file in that directory.

7. Choose whether to activate server-parsed HTML.

You can activate for HTML files but not the `exec` tag, or for HTML files and the `exec` tag, which allows HTML files to execute other programs on the server.

8. Choose which files to parse.

You can choose whether to parse only files with the `.shtml` extension, or all HTML files, which slows performance. If you are using UNIX, you can also choose to parse UNIX files with the `execute` permission turned on, though that can be unreliable.

9. Click on the OK button.
10. Go to the server instance page and select the Apply Changes button.

When you activate parsing, you need to be sure that the following directives are added to your `init.conf` file (note that native threads are turned off):

```
Init funcs="shtml_init,shtml_send" shlib="install_dir/ lib/libShtml.so"
NativeThread="no" fn="load-modules"
```

Note that you must set `NativeThread="no"` for Sun Java System Application Server 7. In addition, these functions now originate from `libShtml.so`, which is located in `install_dir/lib`.

In addition, be sure that the following directive is added to your `obj.conf` file:

```
<Object name="default">
...
...
Service fn="shtml_send" type="magnus-internal/parsed-html" method="(GET|HEAD)"
...
</Object>
```

Using Server-Side HTML Commands

This section describes the HTML commands for including server-parsed tags in HTML files. These commands are embedded into HTML files, which are processed by the `obj.conf` file's `parse-html` function.

The server replaces each command with data determined by the command and its attributes. The format for a command is:

```
<!--#command attribute1 attribute2 <Body>... -->
```

The format for each attribute is a name-value pair such as:

```
name="value"
```

Commands and attribute names should be in lower case.

The commands are “hidden” within HTML comments so they are ignored if not parsed by the server. The standard server-side commands are:

- `config`
- `include`
- `echo`
- `fsize`
- `flastmod`
- `exec`

config

The `config` command initializes the format for other commands.

- The `errmsg` attribute defines a message sent to the client when an error occurs while parsing the file. This error is also logged in the server log file.
- The `timefmt` attribute determines the format of the date for the `lastmod` command. It uses the same format characters as the `util_strftime` function. The default time format is: "%A, %d-%b-%Y %T".

Refer to [“Time Formats” on page 147](#) for details about time formats.

- The `sizefmt` attribute determines the format of the file size for the `filesize` command. It can have one of these values:
 - `bytes` to report file size as a whole number in the format 12,345,678.
 - `abbrev` (the default) to report file size as a number of KB or MB.

Example:

```
<!--#config timefmt="%r %a %b %e, %Y" sizefmt="abbrev"-->
```

This sets the date format to a value such as 08:23:15 AM Wed Apr 15, 1996, and the file size format to the number of KB or MB of characters used by the file.

include

The `include` command inserts a file into the parsed file. You can nest files by including another parsed file, which then includes another file, and so on. The client requesting the parsed document must also have access to the included file if your server uses access control for the directories where they reside.

In Sun Java System Application Server 7, you can use the `include` command with the `virtual` attribute to include a CGI program file. You must also use an `exec` command to execute the CGI program.

- The `virtual` attribute is the URI of a file on the server.
- The `file` attribute is a relative path name from the current directory. It cannot contain elements such as `../` and it cannot be an absolute path.

Example:

```
<!--#include file="bottle.gif"-->
```


echo

The `echo` command inserts the value of an environment variable. The `var` attribute specifies the environment variable to insert. If the variable is not found, “(none)” is inserted. For a list of environment variables, see the section [“Environment Variables in Server-Side HTML Commands” on page 146](#).

Example:

```
<!--#echo var="DATE_GMT"-->
```

fsize

The `fsize` command sends the size of a file. The attributes are the same as those for the `include` command (`virtual` and `file`). The file size format is determined by the `sizefmt` attribute in the `config` command.

Example:

```
<!--#fsize file="bottle.gif"-->
```

flastmod

The `flastmod` command prints the date a file was last modified. The attributes are the same as those for the `include` command (`virtual` and `file`). The date format is determined by the `timefmt` attribute in the `config` command.

Example:

```
<!--#flastmod file="bottle.gif"-->
```

exec

The `exec` command runs a shell command or CGI program.

- The `cmd` attribute (UNIX only) runs a command using `/bin/sh`. You may include any special environment variables in the command.
- The `cgi` attribute runs a CGI program and includes its output in the parsed file.

Example:

```
<!--#exec cgi="workit.pl"-->
```

Environment Variables in Server-Side HTML Commands

In addition to the normal set of environment variables used in CGI, you may include the following variables in your parsed commands:

- `DOCUMENT_NAME`
is the file name of the parsed file.
- `DOCUMENT_URI`
is the virtual path to the parsed file (for example, `/shtml/test.shtml`).
- `QUERY_STRING_UNESCAPED`
is the unescaped version of any search query the client sent with all shell-special characters escaped with the `\` character.
- `DATE_LOCAL`
is the current date and local time.
- `DATE_GMT`
is the current date and time expressed in Greenwich Mean Time.
- `LAST_MODIFIED`
is the date the file was last modified.

Embedding Servlets

Sun Java System Application Server 7 supports the `<SERVLET>` tag as introduced by Java Web Server. This tag allows you to embed servlet output in an SHTML file. No configuration changes are necessary to enable this behavior. If SSI and servlets are both enabled, the `<SERVLET>` tag is enabled.

The `<SERVLET>` tag syntax is slightly different from that of other SSI commands; it resembles the `<APPLET>` tag syntax:

```
<servlet code=code>
<param name=param1 value=v3>
<param name=param2 value=v4>
.
.
</servlet>
```

The `code` parameter specifies the URI of the servlet, including the web application context root. This URI must match a `url-pattern` subelement of a `servlet-mapping` element in the J2EE deployment descriptor (`web.xml`).

Time Formats

The following table describes the format strings for dates and times used by server-parsed HTML.

Table 7-1 Time Formats

Symbol	Meaning
%a	Abbreviated weekday name (3 chars)
%d	Day of month as decimal number (01-31)
%S	Second as decimal number (00-59)
%M	Minute as decimal number (00-59)
%H	Hour in 24-hour format (00-23)
%Y	Year with century, as decimal number, up to 2099
%b	Abbreviated month name (3 chars)
%h	Abbreviated month name (3 chars)
%T	Time "HH:MM:SS"
%X	Time "HH:MM:SS"
%A	Full weekday name
%B	Full month name
%C	"%a %b %e %H:%M:%S %Y"
%c	Date & time "%m/%d/%Y %H:%M:%S"
%D	Date "%m/%d/%Y"
%e	Day of month as decimal number (1-31) without leading zeros
%I	Hour in 12-hour format (01-12)
%j	Day of year as decimal number (001-366)
%k	Hour in 24-hour format (0-23) without leading zeros
%l	Hour in 12-hour format (1-12) without leading zeros
%m	Month as decimal number (01-12)
%n	line feed

Table 7-1 Time Formats (*Continued*)

Symbol	Meaning
%p	A.M./P.M. indicator for 12-hour clock
%R	Time "%H: %M"
%r	Time "%I: %M: %S %p"
%t	tab
%U	Week of year as decimal number, with Sunday as first day of week (00-51)
%w	Weekday as decimal number (0-6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00-51)
%x	Date "%m/ %d/ %Y"
%y	Year without century, as decimal number (00-99)
%%	Percent sign

Using CGI

Common Gateway Interface (CGI) programs run on the server and generate a response to return to the requesting client. CGI programs can be written in various languages, including C, C++, Perl, and as shell scripts. CGI programs are invoked through URL invocation.

A myriad of information about writing CGI programs is available. A good starting point is “The Common Gateway Interface” at:

<http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>

Sun Java System Application Server complies with the version 1.1 CGI specification.

Since the server starts up a process each time the CGI script or program runs, this is an expensive method of programming the server.

For security, CGI scripts depend on the server’s security configuration. For more information, see “[Security for SHTML and CGI](#)” on page 90 and the *Sun Java System Application Server Administrator's Guide to Security*.

This chapter includes the following topics:

- [CGI and J2EE Web Applications](#)
- [Enabling CGI](#)
- [Creating Custom Execution Environments for CGI Programs \(UNIX only\)](#)
- [Adding CGI Programs to the Server](#)
- [Setting the Priority of a CGI Program](#)
- [The Query Handler](#)
- [Perl CGI Programs](#)
- [Global CGI Settings](#)
- [CGI Variables](#)

CGI and J2EE Web Applications

In Sun Java System Application Server, CGI programs cannot interoperate with J2EE web applications. Specifically:

- Do not place CGI programs within web application context roots.
- Do not include the output of CGI programs in servlets or JSPs.
- Do not forward requests to CGI programs from servlets or JSPs.
- You cannot apply J2EE security-constraint and filter-mapping features to CGI programs.

Enabling CGI

Sun Java System Application Server provides these ways to identify CGI programs:

- [Specifying CGI Directories](#). The server treats all files in CGI directories as CGI programs.
- [Specifying CGI File Extensions](#). The server treats all files with the specified extensions as CGI programs.

Specifying CGI Directories

To specify directories that contain CGI programs (and only CGI programs):

1. Create the CGI directory on your computer. This directory doesn't have to be a subdirectory of your document root directory. This is why you must specify a URL prefix in [Step 7](#).
2. Open the HTTP Server component under your server instance in the Administration interface.
3. Go to the Virtual Servers page.
4. Click on the name of the virtual server for which you are specifying a CGI directory.
5. Click on the CGI tab.
6. Click on the CGI Directory option.

7. In the URL Prefix field, type the URL prefix to use for this directory. That is, the text you type appears as the directory for the CGI programs in URLs.

For example, if you type `cgi-bin` as the URL prefix, then all URLs to these CGI programs have the following structure:

`http://yourserver.domain.com/cgi-bin/program-name`

NOTE The URL prefix you specify can be different from the real CGI directory you specify in the previous step.

8. In the CGI Directory text field, type the location of the directory as an absolute path.
9. Click on the OK button.
10. Go to the server instance page and select the Apply Changes button.

The server treats all files in these directories as CGI programs.

To remove an existing CGI directory, click that directory's Remove button in the CGI Directory page. To change the URL prefix or CGI directory of an existing directory, click that directory's Edit button.

Copy your CGI programs into the directories you've specified. Remember that any files in those directories are processed as CGI files, so don't put HTML files in your CGI directory.

For each CGI directory, the file `obj.conf` contains a `NameTrans` directive that associates the name `cgi` with each request for a resource in that directory. These directives are automatically added to `obj.conf` when you specify CGI directories in the Administration interface, or you can manually add them to `obj.conf` if desired.

For example, the following instruction interprets all requests for resources in `http://server-name/cgi-local` as requests to invoke CGI programs in the directory `C:/SunServer/docs/mycgi`.

```
NameTrans fn="pfx2dir" from="/cgi-local" dir="C:/SunServer/docs/mycgi"
name="cgi"
```

The `obj.conf` file must contain the following named object:

```
<Object name="cgi">
  ObjectType fn="force-type" type="magnus-internal/cgi"
  Service fn="send-cgi"
</Object>
```

Do not remove this object from `obj.conf`. If you do, the server will never recognize CGI directories, regardless of whether you specify them in the Administration interface or manually add more `NameTrans` directives to `obj.conf`.

Specifying CGI File Extensions

To instruct the server to treat all files with certain extensions as CGI programs, regardless of which directory they reside in:

1. Open the HTTP Server component under your server instance in the Administration interface.
2. Go to the Virtual Servers page.
3. Click on the name of the virtual server for which you are specifying CGI file types.
4. Click on the CGI tab.
5. Click on the CGI File Type option.
6. From the Editing picker, choose the resource you want this change to apply to.
7. Click the Yes radio button under Activate CGI as a File Type.
8. Click on the OK button.
9. Go to the server instance page and select the Apply Changes button.

The default CGI extensions are `.cgi`, `.bat` and `.exe`.

To change which extensions indicate CGI programs, modify the following line in `mime.types` to specify the desired extensions. Be sure to restart the server after editing `mime.types`.

```
type=magnus-internal/cgi exts=cgi,exe,bat
```

When the server is enabled to treat all files with an appropriate extensions as CGI programs, the `obj.conf` file contains the following Service directive:

```
Service fn="send-cgi" type="magnus-internal/cgi"
```


Creating Custom Execution Environments for CGI Programs (UNIX only)

Before you can create a custom execution environment, you must install the `suid Cgistub` and run it as root:

1. Log in as the superuser.

```
su
```

2. Create the `private` directory for `Cgistub`:

```
cd instance_dir
mkdir private
```

3. Copy `Cgistub` to the `private` directory:

```
cd private
cp install_dir/lib/Cgistub .
```

4. Set the owner of `private` to the server user:

```
chown username .
```

5. Set the permissions on `private`:

```
chmod 500 .
```

6. Set the owner of `Cgistub` to root:

```
chown root Cgistub
```

7. Set the permissions on `Cgistub`:

```
chmod 4711 Cgistub
```

8. You can give each reference to the `send-cgi` function in `obj.conf` a `user` parameter. For example:

```
Service fn="send-cgi" user="username"
```

You can use variable substitution. For example, in `server.xml`, give a `virtual-server` element the following property subelement:

```
<property name="user" value="username" />
```

This lets you write the `send-cgi` function line in `obj.conf` as follows:

```
Service fn="send-cgi" user="$user"
```

For more information about `send-cgi` and `obj.conf`, see the *Sun Java System Application Server Developer's Guide to NSAPI*. For more information about `server.xml`, see the *Sun Java System Application Server Administrator's Configuration File Reference*.

9. Restart the server so these changes take effect.

NOTE You should install `Cgistub` in the `instance_dir/private` directory. If you install it anywhere else, you must specify the path to `Cgistub` in the `init-cgi` function in `init.conf`. For details, see the *Sun Java System Application Server Developer's Guide to NSAPI*.

NOTE It may not be possible to install the `suid Cgistub` program on an NFS mount. If you wish to use an `suid Cgistub`, you must install your server instance to a local file system.

`Cgistub` enforces the following security restrictions:

- The user the CGI program executes as must have a `uid` of 100 or greater. This prevents anyone from using `Cgistub` to obtain root access.
- The CGI program must be owned by the user it is executed as and must not be writable by anyone other than its owner. This makes it difficult for anyone to covertly inject and then remotely execute programs.
- `Cgistub` creates its UNIX listen socket with 0700 permissions.

NOTE Socket permissions are not respected on a number of UNIX variants, including current versions of SunOS/Solaris. To prevent a malicious user from exploiting `Cgistub`, change the server's temporary directory (using the `init.conf TempDir` directive) to a directory accessible only to the server user. For details, see the *Sun Java System Application Server Administrator's Configuration File Reference*.

After you have installed `Cgistub`, you can create custom execution environments in the following ways:

- [Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server](#)
- [Specifying a Chroot Directory for a Virtual Server](#)

Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server

To prevent a virtual server's CGI programs from interfering with other users, these programs should be stored in a unique directory and execute with the permissions of a unique UNIX user and group.

First, create the UNIX user and group. The exact steps required to create a user and group vary by operating system. For help, consult your operating system's documentation.

Next, follow these steps to create a `cgi-bin` directory for the virtual server:

1. Log in as the superuser.

```
su
```

2. Change to the virtual server directory.

```
cd vs_dir
```

3. Create the `cgi-bin` directory.

```
mkdir cgi-bin
```

```
chown user:group cgi-bin
```

```
chmod 755 cgi-bin
```

You can set the virtual server's CGI directory, user, and group in one of these ways:

- Use the `dir`, `user`, and `group` parameters of the `send-cgi` function in the `obj.conf` file. For more information, see the *Sun Java System Application Server Developer's Guide to NSAPI*.
- Enter this information using the Administration interface:
 - a. Open the HTTP Server component under your server instance.
 - b. Go to the Virtual Servers page.
 - c. Click on the name of the virtual server for which you are specifying CGI directories.
 - d. Click on the General tab.
 - e. Type values in the Directory, User, and Group fields.
 - f. Click on the Save button.
 - g. Go to the server instance page and select the Apply Changes button.

For more information, see the *Sun Java System Application Server Administrator's Guide*.

Specifying a Chroot Directory for a Virtual Server

To further improve security, these CGI scripts should be prevented from accessing data above and outside of the virtual server directory.

First, set up the chroot environment. The exact steps required to set up the chroot environment vary by operating system. For help, consult your operating system's documentation. The man pages for `ftpd` and `chroot` are often a good place to start.

These are the steps required for Solaris versions 2.6 through 8:

1. Log in as the superuser.

```
su
```

2. Change to the chroot directory. This is typically the `vs_dir` directory mentioned in the previous section.

```
cd chroot
```

3. Create `tmp` in the chroot directory:

```
mkdir tmp
```

```
chmod 1777 tmp
```

4. Create `dev` in the chroot directory:

```
mkdir dev
```

```
chmod 755 dev
```

5. List `/dev/tcp`, and note the major and minor numbers of the resulting output. In this example, the major number is 11 and the minor number is 42:

```
ls -lL /dev/tcp
```

```
crw-rw-rw-  1 root      sys      11, 42 Apr  9 1998 /dev/tcp
```

6. Create the `tcp` device using the major and minor numbers:

```
mknod dev/tcp c 11 42
```

```
chmod 666 dev/tcp
```

7. Repeat steps 5 and 6 for each of the following devices (each device will have a different major and minor combination):

```
/dev/udp
/dev/ip
/dev/kmem
/dev/kstat
/dev/ksyms
/dev/mem
/dev/null
/dev/stderr
/dev/stdin
/dev/stdout
/dev/ticotsord
/dev/zero
```

8. Set permissions on the devices in `dev` in the `chroot` directory:

```
chmod 666 dev/*
```

9. Create and populate `usr/lib` and `usr/lib` in the `chroot` directory:

```
mkdir usr
mkdir usr/lib
ln -s /usr/lib
ln /usr/lib/* usr/lib
```

You can ignore the messages this command generates.

If the `/usr/lib` directory is on a different file system, replace the last command with the following:

```
cp -rf /usr/lib/* usr/lib
```

10. Create and populate `bin` and `usr/bin` in the `chroot` directory:

```
mkdir usr/bin
ln -s /usr/bin
ln /usr/bin/* usr/bin
```

You can ignore the messages this command generates.

If the `/usr/bin` directory is on a different file system, replace the last command with the following:

```
cp -rf /usr/bin/* usr/bin
```

11. Create and populate etc in the chroot directory:

```
mkdir etc

ln /etc/passwd /etc/group /etc/netconfig etc
```

12. Test the chroot environment:

```
chroot chroot bin/ls -l
```

The output should look something like this:

```
total 14
lrwxrwxrwx    1 root    other      8 Jan 13 03:32 bin -> /usr/bin
drwxr-xr-x    2 user    group      512 Jan 13 03:42 cgi-bin
drwxr-xr-x    2 root    other      512 Jan 13 03:28 dev
drwxr-xr-x    2 user    group      512 Jan 13 03:26 docs
drwxr-xr-x    2 root    other      512 Jan 13 03:33 etc
lrwxrwxrwx    1 root    other      8 Jan 13 03:30 lib -> /usr/lib
drwxr-xr-x    4 root    other      512 Jan 13 03:32 usr
```

You can set the virtual server's chroot directory in one of these ways:

- Use the `chroot` parameter of the `send-cgi` function in the `obj.conf` file. For more information, see the *Sun Java System Application Server Developer's Guide to NSAPI*.
- Enter this information using the Administration interface:
 - a. Open the HTTP Server component under your server instance.
 - b. Go to the Virtual Servers page.
 - c. Click on the name of the virtual server for which you are specifying CGI directories.
 - d. Click on the General tab.
 - e. Type a value in the Chroot field.
 - f. Click on the Save button.
 - g. Go to the server instance page and select the Apply Changes button.

For more information, see the *Sun Java System Application Server Administrator's Guide*.

Adding CGI Programs to the Server

To add CGI programs to the Sun Java System Application Server, simply do one of the following:

- Drop the program file in a CGI directory (if there are any).
- Give it a file name that the server recognizes as a CGI program and put it in any directory at or below the document root (if CGI file type recognition has been activated).

For UNIX, make sure the program file has execute permissions set.

Setting the Priority of a CGI Program

To set the priority of a CGI program:

1. Open the HTTP Server component under your server instance in the Administration interface.
2. Go to the Virtual Servers page.
3. Click on the name of the virtual server for which you are specifying CGI directories.
4. Click on the General tab.
5. Type a value in the Nice field. This increment determines the CGI program's priority relative to the server. Typically, the server is run with a nice value of 0 and the nice increment would be between 0 (the CGI program runs at same priority as server) and 19 (the CGI program runs at much lower priority than server). You should not increase the priority of the CGI program above that of the server.
6. Click on the Save button.
7. Go to the server instance page and select the Apply Changes button.

For more information, see the *Sun Java System Application Server Administrator's Guide*.

The Query Handler

NOTE

The use of Query Handlers is outdated. Although Sun Java System Application Server and Netscape Navigator clients still support it, it is rarely used. It is much more common for people to use forms in their HTML pages to submit queries.

You can specify a default query handler CGI program. A query handler processes text sent to it via the ISINDEX tag in an HTML file.

ISINDEX is similar to a form text field in that it creates a text field in the HTML page that can accept typed input. Unlike the information in a form text field, however, the information in the ISINDEX box is immediately submitted when the user presses Return. When you specify your default query handler, you tell your server to which program to direct the input. For an in-depth discussion of the ISINDEX tag, see an HTML reference manual.

To set a query handler, perform the following steps:

1. Open the HTTP Server component under your server instance in the Administration interface.
2. Go to the Virtual Servers page.
3. Click on the name of the virtual server for which you are specifying a query handler.
4. Click on the CGI tab.
5. Click on the Query Handler option.
6. Use the Editing Picker to select the resource you want to set with a default query handler.

If you choose a directory, the query handler you specify runs only when the server receives a URL for that directory or any file in that directory.

7. In the Default Query Handler field, enter the full path for the CGI program you want to use as the default for the resource you chose.
8. Click on the OK button.
9. Go to the server instance page and select the Apply Changes button.

Perl CGI Programs

You cannot run CGIs using Perl 5.6.x with the `-w` flag. Instead, include the following code in the file:

```
use warnings;
```


Global CGI Settings

To change global CGI settings:

1. Open the HTTP Server component under your server instance in the Administration interface.
2. Go to the HTTP Server page.
3. Click on the Advanced tab.
4. Click on the CGI option.
5. You can change the following settings:
 - `MinCGIStubs` - Sets the number of CGIStub processes that are started by default. This value must be lower than `MaxCGIStubs`. The default is 2.
 - `CGIExpirationTimeout` - Specifies the maximum time in seconds that CGI processes are allowed to run before being killed. The default is 0, which means processes are allowed to run indefinitely.
 - `CGIStubIdleTimeout` - Kills any CGIStub processes that have been idle for this number of seconds. The default is 30.
 - `MaxCGIStubs` - Sets the maximum number of CGIStub processes the server can execute concurrently. The default is 10.
6. Click on the OK button.
7. Go to the server instance page and select the Apply Changes button.

For more information about these global CGI settings, see the `init.conf` file description in the *Sun Java System Application Server Administrator's Configuration File Reference*.

CGI Variables

In addition to the standard CGI variables, you can use the Sun Java System Application Server CGI variables in CGI programs to access information about the client certificate if the server is running in secure mode. The `CLIENT_CERT` and `REVOCATION` variables are available only when client certificate based authentication is enabled.

The following table lists the Sun Java System Application Server CGI variables.

Table 8-1 CGI Variables

Variable	Description
SERVER_URL	The URL of the server that the client requested
HTTP_xxx	An incoming HTTP request header, where xxx is the name of the header
HTTPS	ON if the server is in secure mode and OFF otherwise
HTTPS_KEYSIZE	The keysize of the SSL handshake (available if the server is in secure mode)
HTTPS_SECRETKEYSIZE	The keysize of the secret part of the SSL handshake (available if the server is in secure mode)
HTTPS_SESSIONID	The session ID for the connection (available if the server is in secure mode)
CLIENT_CERT	The certificate that the client provided (binary DER format)
CLIENT_CERT_SUBJECT_DN	The Distinguished Name of the subject of the client certificate
CLIENT_CERT_SUBJECT_OU	The Organization Unit of the subject of the client certificate
CLIENT_CERT_SUBJECT_O	The Organization of the subject of the client certificate
CLIENT_CERT_SUBJECT_C	The Country of the subject of the client certificate
CLIENT_CERT_SUBJECT_L	The Location of the subject of the client certificate
CLIENT_CERT_SUBJECT_ST	The State of the subject of the client certificate
CLIENT_CERT_SUBJECT_E	The E-mail of the subject of the client certificate
CLIENT_CERT_SUBJECT_UID	The UID part of the CN of the subject of the client certificate
CLIENT_CERT_ISSUER_DN	The Distinguished Name of the issuer of the client certificate
CLIENT_CERT_ISSUER_OU	The Organization Unit of the issuer of the client certificate
CLIENT_CERT_ISSUER_O	The Organization of the issuer of the client certificate
CLIENT_CERT_ISSUER_C	The Country of the issuer of the client certificate
CLIENT_CERT_ISSUER_L	The Location of the issuer of the client certificate
CLIENT_CERT_ISSUER_ST	The State of the issuer of the client certificate
CLIENT_CERT_ISSUER_E	The E-mail of the issuer of the client certificate

Table 8-1 CGI Variables (*Continued*)

Variable	Description
CLIENT_CERT_ISSUER_UID	The UID part of the CN of the issuer of the client certificate
CLIENT_CERT_VALIDITY_START	The start date of the certificate
CLIENT_CERT_VALIDITY_EXPIRES	The expiration date of the certificate
CLIENT_CERT_EXTENSION_xxx	The certificate extension, where xxx is the name of the extension
REVOCATION_METHOD	The name of the certificate revocation method if it exists
REVOCATION_STATUS	The status of certificate revocation if it exists

Index

A

- abbrev, value of sizefmt attribute [144](#)
- Administration interface
 - changing servlet output [46](#)
 - configuring a default web module [27](#)
 - configuring the web container [29](#)
 - deployment using [97](#)
 - enabling SHTML [142](#)
 - global CGI settings [161](#)
 - setting a query handler [160](#)
 - setting CGI program priority [159](#)
 - setting the default locale [25](#)
 - setting the virtual server's CGI directory [155](#)
 - setting the virtual server's chroot directory [158](#)
 - setting up dynamic reloading [99](#)
 - specifying CGI directories [150](#)
 - specifying CGI file extensions [152](#)
- agent attribute [133](#)
- API reference
 - CGI [149](#)
 - JavaBeans [23](#)
 - JSP [23](#), [57](#)
 - servlets [22](#)
- appserv-tags.jar file [58](#)
- appserv-tags.tld file [58](#)
- asadmin deploy command [61](#), [97](#)
- attributes, about [101](#)
- authentication [85](#)
 - form-based login [87](#)
 - HTTP basic [86](#)
 - single sign-on [87](#)
 - SSL mutual [86](#)

- authorization [89](#)
 - constraints [90](#)
 - roles [89](#)
- Availability Service, enabling [74](#)

B

- BaseCache cacheClassName value [121](#)
- basic authentication [86](#)
- beans in JSPs [23](#)
- bin directory for CGI [157](#)
- BoundedMultiLruCache cacheClassName value [122](#)
- bytes, value of sizefmt attribute [144](#)

C

- cache [47](#), [120](#)
 - default configuration [49](#)
 - example configuration [49](#)
 - for JSPs [58](#)
 - for static file content [47](#)
 - helper class [48](#), [52](#)
- cache element [120](#)
- cache tag [58](#)
- cacheClassName property [121](#)
- cache-helper element [122](#)
- CacheHelper interface [50](#), [52](#), [122](#)
- cache-helper-ref element [125](#)

- cacheKeyGeneratorAttrName property 52, 123
- cache-mapping element 123
- cache-on-match attribute 127, 128
- cache-on-match-failure attribute 128
- certificate
 - CGI variables 161
 - fetching 90
- CGI 149
 - adding programs to the server 159
 - and J2EE applications 150
 - and virtual servers 155
 - client certificate variables 161
 - custom execution environment 153
 - enabling 150
 - global settings 161
 - Perl programs 160
 - security for 90
 - setting a program's priority 159
 - specifying CGI directories 150
 - specifying file extensions 152
 - variables 161
 - website 149
- cgi attribute of exec command 145
- cgi-bin directory 155
- CGIExpirationTimeout CGI setting 161
- Cgistub 153
- CGIStubIdleTimeout CGI setting 161
- charset attribute 133
- chroot directory 156
- cladmin command 98
- classdebuginfo attribute 131
- classes directory 94
- classloader delegation model 129
- class-loader element 129
- class-name attribute 122
- client certificate
 - CGI variables 161
 - fetching 90
- CloudScape 29
- cluster, deployment to 98
- cmd attribute of exec command 145
- compiling JSPs 61
- config SHTML command 144
- connection pooling, database 29

- constraint-field element 127
- contentType attribute of page directive 26
- cookieComment property 115
- cookieDomain property 114
- cookieMaxAgeSeconds property 114
- cookiePath property 114
- cookie-properties element 114
- cookies 66
- crossContextAllowed property 104
- custom execution environment 153

D

- database connection pooling 29
- DATE_GMT SHTML variable 146
- DATE_LOCAL SHTML variable 146
- debugging 24
- default virtual server 27
- default web module 28, 44
- default-helper element 122
- default-locale attribute 132
- default-resource-principal element 117
- delegate attribute 129
- deployment 93
 - approaches 96
 - descriptors 95
 - dynamic 96
 - dynamic reloading 99
 - to a cluster 98
- description element 105
- destroy method 36, 53
- destroying servlets 53
- directory property 83, 112
- distributable web application 67
- distributed session 67
- DOCUMENT_NAME SHTML variable 146
- DOCUMENT_URI SHTML variable 146
- doGet method 34, 37, 53, 54
- doPost method 34, 37, 53, 54
- dynamic
 - reloading of JSPs 61, 100

reloading of web applications [99](#)
 dynamic deployment [96](#)

E

echo SHTML command [145](#)
 EJB components, accessing [39](#)
 ejb-ref element [118](#)
 ejb-ref-name element [119](#)
 enableCookies property [113](#)
 enabled attribute [121](#)
 enableURLRewriting property [114](#)
 encoding
 hidden field for [133](#)
 of JSPs [26](#), [131](#)
 of servlets [25](#)
 environment variables, SHTML [146](#)
 errmsg attribute of config command [144](#)
 exceptions [57](#)
 exec SHTML command [145](#)
 execution environment [153](#)
 extensions, for CGI [152](#)
 extra-class-path attribute [129](#)

F

failover, object types supported for [68](#)
 field, hidden, for character encoding [133](#)
 file attribute of include command [144](#)
 file extensions, for CGI [152](#)
 flastmod SHTML command [145](#)
 affected by timefmt attribute [144](#)
 flush tag [60](#)
 formats, time [147](#)
 form-based login [87](#)
 form-hint-field attribute [134](#)
 forward method [43](#), [45](#)
 fsize SHTML command [145](#)

G

generic servlets [34](#)
 getAttribute method [73](#)
 getAttributeNames method [73](#)
 getCreationTime method [71](#)
 getId method [71](#)
 getLastAccessedTime method [71](#)
 getMethod method [37](#), [54](#)
 getParameter method [133](#)
 getRemoteUser method [72](#)
 getRequestedSessionId method [72](#)
 getSession method [70](#)
 group-name element [107](#)
 groups in realms [106](#)

H

HADB (high-availability database) [80](#)
 Handler, Query [159](#)
 handling requests [53](#)
 hidden field, for character encoding [133](#)
 high availability
 configuring [80](#)
 enabling [74](#)
 HTML tags, server-parsed commands *see* SHTML
 HTTP basic authentication [86](#)
 HTTP servlets [34](#)
 http-method element [126](#)
 HttpServletRequest [48](#), [70](#)
 HttpSession [71](#)
 HttpSession interface [65](#)
 HttpSessionBindingListener interface [73](#)

I

idLengthBytes property [114](#)
 ieClassId property [130](#)
 include method [43](#)

- include SHTML command [144](#)
- init method [36, 53](#)
- init.conf file, and CGI [154, 161](#)
- init-cgi function [154](#)
- instantiating servlets [53](#)
- internationalization [25, 132](#)
- invalidate method [73](#)
- ISINDEX tag [160](#)
- isNew method [71](#)
- isRequestedSessionIdFromCookie method [72](#)
- isRequestedSessionIdFromURL method [72](#)
- isRequestedSessionIdValid method [72](#)

J

- Java Database Connectivity *see* JDBC
- Java Message Service *see* JMS
- Java Naming Directory Interface *see* JNDI
- Java Servlet API [22](#)
- JavaBeans [23](#)
- javaCompilerPath property [130](#)
- javaCompilerPlugin property [130](#)
- javaEncoding attribute [131](#)
- JDBC driver [29](#)
- jikes compiler [130](#)
- JMS [117](#)
- JNDI [40, 41](#)
- jndi-name element [119](#)
- JSP 1.2 specification [23, 57](#)
- jspc command [61](#)
- jsp-config element [130](#)
- JSPs
 - about [22, 55](#)
 - accessing Java functionality [22](#)
 - API reference [23, 57](#)
 - beans in [23](#)
 - caching [58](#)
 - character encoding [26](#)
 - command-line compiler [61](#)
 - compiler for [130](#)
 - configuring [130](#)

- creating [56](#)
- debugging [64](#)
- dynamic reloading of [61, 100, 131](#)
- encoding of [131](#)
- exceptions [57](#)
- generated source code [96](#)
- invoking from servlets [43](#)
- portability [57](#)
- precompiling [61](#)
- syntax [56](#)
- tag libraries [57](#)

K

- keepgenerated property [131](#)
- key attribute
 - of cache tag [59](#)
 - of flush tag [60](#)
- key-field element [126](#)

L

- largefile property [131](#)
- LAST_MODIFIED SHTML variable [146](#)
- lib directory
 - and JSP tags [58](#)
 - for a web application [94](#)
 - for CGI [157](#)
 - for the entire server [100, 143](#)
- load balancing [98](#)
- locale attribute
 - server.xml file [25](#)
 - sun-web.xml file [133](#)
- locale-charset-info element [132](#)
- locale-charset-map element [132](#)
- logging in the web container [29](#)
- login, form-based [87](#)
- LruCache cacheClassName value [121](#)

M

- manager-properties element [110](#)
- mappedfile property [131](#)
- match-expr attribute [128](#)
- MaxCGIStubs CGI setting [161](#)
- max-entries attribute [120](#)
- maxSessions property [80](#), [82](#), [110](#)
- MaxSize property [121](#)
- mime.types file, CGI extensions in [152](#)
- MinCGIStubs CGI setting [161](#)
- MultiLruCache cacheClassName value [121](#)
- MultiLRUSegmentSize property [121](#)
- mutual authentication [86](#)

N

- name attribute [105](#), [122](#), [125](#), [126](#), [127](#)
- name element [118](#)
- nocache attribute of cache tag [59](#)
- nsfc.conf file [47](#)

O

- obj.conf file
 - and CGI [151](#), [152](#), [153](#), [155](#), [158](#)
 - and SHTML [143](#)
- Oracle [29](#)
- output from servlets [46](#)

P

- page directive [26](#)
- parameter-encoding element [133](#)
- parameters, servlet, accessing [38](#)
- parse-html function [143](#)
- password element [118](#)
- Perl [160](#)

- persistenceFrequency property [82](#), [111](#)
- persistenceScope property [83](#), [112](#)
- persistence-type attribute [109](#)
- persistent session manger [80](#)
- PersistentManager [80](#)
- plugin tag [130](#)
- Pointbase [29](#)
- pooling
 - of database connections [29](#)
 - of servlets [54](#)
- portability [57](#)
- precompiling JSPs [61](#)
- principal-name element [107](#)
- private directory [154](#)
- programmatic login [86](#)
- properties, about [104](#)
- property element [104](#)

Q

- Query Handler [159](#)
- QUERY_STRING_UNESCAPED SHTML variable [146](#)

R

- realms
 - mapping groups and users to [106](#)
 - use in form-based login [87](#)
- reapIntervalSeconds property [79](#), [82](#), [110](#)
- refresh attribute of cache tag [59](#)
- refresh-field element [125](#)
- .reload file [99](#)
- reloading, dynamic [99](#)
 - of JSPs [61](#), [100](#)
- reload-interval property [131](#)
- removeAttribute method [73](#)
- removing servlets [53](#)
- request object [53](#)
- requirement rules [100](#)

- resource allocation 54
- resource-env-ref element 115
- resource-env-ref-name element 116
- resource-ref element 116
- response pages 43
- res-ref-name element 117
- role-name element 107
- roles 89
- rpm 16

S

- scope attribute 125, 126, 127
- scratchdir property 131
- Secure Socket Layer *see* SSL
- security 85
 - and servlets 39
 - of sessions 67
- security-role-mapping element 106
- send-cgi function 153, 155, 158
- server.xml file
 - and JNDI names 119
 - changing servlet output 47
 - configuring a default web module 28
 - configuring single sign-on 88
 - configuring the web container 29
 - setting availability 76
 - setting the default locale 25
 - setting up dynamic reloading 99
 - variables in 153
- server-parsed HTML *see* SHTML
- service method 34, 37, 53, 54
- Servlet 2.3 specification 22
- servlet element 106
- <SERVLET> tag 146
- servlet-name element 107
- servlets
 - about 22, 31
 - accessing EJB components 39
 - accessing parameters 38
 - API reference 22
 - caching 47
 - character encoding 25
 - creating 34
 - destroying 53
 - embedding in HTML files 146
 - engine 53, 54
 - execution cycle 33
 - generic vs. HTTP 34
 - instantiating 53
 - invoking from another servlet 45
 - invoking JSPs from 43
 - invoking using a URL 44
 - output 46
 - pooling 54
 - removing 53
 - request handling 53
 - response pages 43
 - security 39
 - sessions 39
 - specification 22
 - thread safety 41
 - user authentication 85
 - user authorization 89
- session managers 108
 - default 78
 - persistent 80
 - PersistentManager 80
 - StandardManager 78
- session-config element 108
- sessionFilename property 80, 111
- session-manager element 109
- session-properties element 113
- sessions
 - about 65
 - cookies 66
 - distributed 67
 - invalidating 73
 - object types supported for failover 68
 - properties 71
 - security 38, 67
 - servlets 39
 - session managers 78
 - URL rewriting 66
- session-timeout element 113
- setAttribute method 39, 73
- setCharacterEncoding method 25
- setContentType method 26, 27

- setLocale method 26
- showrev 16
- SHTML 141
 - and J2EE applications 141
 - commands and syntax 143
 - embedding servlets in 146
 - enabling 142
 - environment variables 146
 - security for 90
- single sign-on 78, 87
- singleThreadedServletPoolSize property 42, 54, 104
- SingleThreadModel class 42
- sizefmt attribute of config command 144
- SSI 146
- SSL
 - and CGI 162
 - mutual authentication 86
- StandardManager 78
- store-pool-jndi-name property 75, 76
- store-properties element 111
- subelements, about 100
- Sun customer support 16
- Sun Java System Message Queue 117
- Sun ONE Studio
 - creating web applications using 24
 - debugging JSPs using 64
 - deployment using 98
- sun-appserv-jspc Ant task 61
- sun-web.xml file
 - elements in 102
 - example of 140
 - schema for 100
- sun-web-app element 103
- sun-web-app_2_3-0.dtd file 100
- syntax of JSPs 56

T

- tag libraries 57
- tags
 - for JSP caching 58
 - SHTML 141

- tempdir property 104
- thread safety 41
- time formats 147
- timefmt attribute of config command 144
- timeout attribute of cache tag 59
- timeout element 125
- timeout-in-seconds attribute 120
- timeoutSeconds property 113

U

- URL rewriting 66
- url-pattern element 124
- useResponseCTForHeaders property 26, 104
- users in realms 106

V

- value attribute 105
- value element 128
- variables
 - CGI 161
 - for send-cgi function 153
 - SHTML 146
- virtual attribute of include command 144
- virtual servers 27
 - and CGI 155
 - default 27

W

- WAR file 20, 94
 - creating 97
- web applications 20
 - creating 23
 - debugging 24
 - deploying 93
 - directory structure 94

Section W

- distributable [67](#)
 - examples [29](#)
- web container, configuring [29](#)
- web module, default [28](#), [44](#)
- web service [129](#)
- web.xml file
 - and the sun-web.xml file [95](#)
 - example of [134](#)
 - location [94](#)
 - session-timeout element [113](#)
- WEB-INF directory [94](#)