Administrator's Guide

Sun™ ONE Application Server

Version 7

817-3652-10
September 2003

Contents

About This Guide

This guide describes how to configure and administer Sun™ ONE Application Server 7. It is intended for information technology administrators in the corporate enterprise who want to extend client-server applications to a broader audience through the World Wide Web.

This preface includes the following sections:

- What’s In This Guide?
- How This Guide Is Organized
- Documentation Conventions
- Product Line Overview
- Using the Documentation
- Product Support

What’s In This Guide?

This guide explains how to configure and administer the Sun ONE Application Server. After configuring your server, use this guide to help maintain your server.

How This Guide Is Organized

This guide is divided into four parts, plus a comprehensive index. Begin with "Part 1, "Server Basics and Administering Global Settings," for an overview of the product. "Part 2, "Managing an Individual Server Instance," introduces you to using the Administration Server, and to using other server functions that affect all server instances.

Once you are familiar with the fundamentals of using the Administration Server, you can refer to "Part 3, "Managing HTTP Server Features and Virtual Servers," which provides information for using programs and configuration styles.

Finally, Appendixes addresses specific reference topics that describe the various topics, including internationalization issues, server extensions, and the Sun ONE Application Server command line interface documentation.

Part I: Server Basics and Administering Global Settings

This part provides an overview of the Sun ONE Application Server. The following chapters are included:

- Chapter 1, "Getting Started with Sun ONE Application Server Administration," provides an overview of Sun ONE Application Server.
- Chapter 2, "Setting Administration Server Preferences," describes how to manage your Administration Server.
- Chapter 3, "Configuring Administrative Domains," describes how to use multiple domains.

Part II: Managing an Individual Server Instance

This part provides conceptual and procedural details about configuring, managing, and using server instances. The following chapters are included:

- Chapter 4, "Using Application Server Instances," describes how to configure server preferences for your Sun ONE Application Server.
- Chapter 5, "Using Logging," describes the foundation for logging, and the logging features and functions within Sun ONE Application Server.

- Chapter 6, “Monitoring the Sun ONE Application Server,” contains information about the monitoring and Simple Network Management Protocol (SNMP) features and functions available within Sun ONE Application Server.
- Chapter 7, “Configuring the Web Server Plugin,” explains how Sun ONE Application Server processes HTTP requests, and how to configure and use the web server plugin with Sun ONE Application Server.
- Chapter 8, “Configuring J2EE Containers,” explains how to configure and use the container that provide runtime support for J2EE application components such as Enterprise Java Beans (EJBs) and Message Driven Beans (MDBs).
- Chapter 9, “Using Transaction Services,” explains database transactions and how to use manage them.
- Chapter 10, “Configuring Naming and Resources,” explains how to configure J2EE resources.
- Chapter 11, “Using the JMS Service,” provides information needed to understand and administer the built-in JMS Service provided through Sun ONE Message Queue, the native JMS provider.
- Chapter 12, “Configuring the Server For CORBA/IIOP Clients,” explains how to configure support for CORBA-based clients, using the RMI/IIOP protocol within an Sun ONE Application Server environment.
- Chapter 13, “Deploying Applications,” describes how to deploy applications to the Sun ONE Application Server.

Part III: Managing HTTP Server Features and Virtual Servers

This part provides information for using the Administration interface to programs and configuration styles. The following chapters are included:

- Chapter 14, “Configuring HTTP Features,” describes how to configure preferences for your HTTP-related features of your Sun ONE Application Server.
- Chapter 15, “Using Virtual Servers,” describes how to set up and administer virtual servers using your Sun ONE Application Server.
- Chapter 16, “Managing Virtual Server Content,” describes how you can configure and manage your server’s content.

Part IV: Appendixes

This section includes various appendixes with reference material that you may wish to review. This section includes the following appendixes:

- Appendix A, “Using the Command Line Interface,” provides instructions for using command line utilities in place of the user interface screens.
- Appendix B, “Third Party Copyright Notices,” contains additional copyright information.

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 21.

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

  - Solaris 8 non-package-based Evaluation installations:

    user's home directory / `sun/appserver7`

  - Solaris unbundled, non-evaluation installations:

    `/opt/SUNWappserver7`

  - Windows, all installations:

    `C:\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 21 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 8 and 9 package-based installation and the Solaris 9 bundled installation, the application server files are spread across several root directories. These directories are described in this section.

- **For Solaris 9 bundled installations**, this guide uses the following document conventions to correspond to the various default installation directories provided:

  - *install_dir* refers to `/usr/appserver/`, 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/appserver/domains`, which is the default location for any domains that are created.
  - *install_config_dir* refers to `/etc/appserver/config`, which contains installation-wide configuration information such as licenses and the master list of administrative domains configured for this installation.

- **For Solaris 8 and 9 package-based, non-evaluation, unbundled installations**, 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.

Product Line Overview

Sun ONE Application Server 7 is a J2EE 1.3 specification-compatible application server which also supports emerging Java Web Services standards as well as standard HTTP server programming facilities. Three editions of the application server are offered to suit a variety of needs for both production and development environments:

- Platform Edition
- Standard Edition
- Enterprise Edition

Platform Edition

Platform Edition forms the core of the Sun ONE Application Server 7 product line. This free-for-production-use product offers a high-performance, small-footprint J2EE 1.3 specification-compatible runtime environment that is ideally suited for basic operational deployments, as well as for embedding in third-party applications. Web-services ready, the Platform Edition includes built-in technologies proven by the Sun ONE Web Server and Sun ONE Message Queue products.

Platform Edition deployments are limited to single application server instances (i.e. single virtual machines for the Java platform ("Java virtual machine" or "JVM™")). Multi-tier deployment topologies are supported by the Platform edition, but the web server tier proxy does not perform load balancing. In Platform Edition, administrative utilities are limited to local clients only.

Platform Edition is integrated into Solaris 9.

Standard Edition

This is the edition that is the focus of this *Administrator's Guide*. The Standard Edition layers enhanced, remote-management capabilities on top of the Platform Edition. Enhanced management capabilities, remote command-line, and web-based administration are all included as part of the Standard Edition. This edition also includes the ability to partition web application traffic through a web server tier proxy. Standard Edition supports configuration of multiple application server instances (JVMs) per machine.

Enterprise Edition

Enterprise Edition enhances the core application server platform with greater high availability, load balancing and clustering capabilities suited for the most demanding J2EE-based application deployments. The management capabilities of the Standard Edition are extended in Enterprise Edition to account for multi-instance and multi-machine deployments.

Clustering support includes easy-to-configure groups of cloned application server instances to which client requests can be load balanced. Both external load balancers and load balancing web tier-based proxies are supported by this edition. HTTP session, stateful session bean instance and Java Message Service (JMS) resource failover are included in the Enterprise Edition. The patented "Always On" highly available database technology forms the basis for the HA persistence store in the Enterprise Edition.

For more product information, see the Sun ONE Application Server page on the Sun Microsystems web site.

Using the Documentation

The Sun ONE Application Server manuals are available as online files in Portable Document Format (PDF) and Hypertext Markup Language (HTML) formats, at:

`http://docs.sun.com/`

The following table lists tasks and concepts described in the Sun ONE Application Server manuals. The left column lists the tasks and concepts, and the right column lists the corresponding manuals.

Table 1 Sun ONE Application Server Documentation Roadmap

For information about	See the following
Late-breaking information about the software and the documentation	Release Notes
Supported platforms and environments	Platform Summary
Introduction to the application server, including new features, evaluation installation information, and architectural overview.	*Getting Started Guide*
Installing Sun ONE Application Server and its various components (sample applications, Administration interface, Sun ONE Message Queue).	*Installation Guide*
Creating and implementing J2EE applications that follow the open Java standards model on the Sun ONE Application Server 7. Includes general information about application design, developer tools, security, assembly, deployment, debugging, and creating lifecycle modules.	*Developer's Guide*

Table 1 Sun ONE Application Server Documentation Roadmap *(Continued)*

For information about	See the following
Creating and implementing J2EE applications that follow the open Java standards model for web applications on the Sun ONE Application Server 7. Discusses web application programming concepts and tasks, and provides sample code, implementation tips, and reference material.	*Developer's Guide to Web Applications*
Creating and implementing J2EE applications that follow the open Java standards model for enterprise beans on the Sun ONE Application Server 7. Discusses EJB programming concepts and tasks, and provides sample code, implementation tips, and reference material.	*Developer's Guide to Enterprise JavaBeans Technology*
Creating clients that access J2EE applications on the Sun ONE Application Server 7	*Developer's Guide to Clients*
Creating web services	*Developer's Guide to Web Services*
J2EE features such as JDBC, JNDI, JTS, JMS, JavaMail, resources, and connectors	*Developer's Guide to J2EE Features and Services*
Creating custom NSAPI plugins	*Developer's Guide to NSAPI*
Performing the following administration tasks: • Using the Administration interface and the command line interface • Configuring server preferences • Using administrative domains • Using server instances • Monitoring and logging server activity • Configuring the web server plugin • Configuring the Java Messaging Service • Using J2EE features • Configuring support for CORBA-based clients • Configuring database connectivity • Configuring transaction management • Configuring the web container • Deploying applications • Managing virtual servers	*Administrator's Guide*

Table 1 Sun ONE Application Server Documentation Roadmap *(Continued)*

For information about	See the following
Editing server configuration files	*Administrator's Configuration File Reference*
Configuring and administering security for the Sun ONE Application Server 7 operational environment. Includes information on general security, certificates, and SSL/TLS encryption. HTTP server-based security is also addressed.	*Administrator's Guide to Security*
Configuring and administering service provider implementation for J2EE CA connectors for the Sun ONE Application Server 7. Includes information about the Administration Tool, DTDs and provides sample XML files.	J2EE CA Service Provider Implementation Administrator's Guide
Migrating your applications to the new Sun ONE Application Server 7 programming model from the Netscape Application Server version 2.1, including a sample migration of an Online Bank application provided with Sun ONE Application Server	*Migrating and Redeploying Server Applications Guide*
Using Sun ONE Message Queue.	The Sun ONE Message Queue documentation at: http://docs.sun.com/

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/supportraining/`

- 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

- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps

Part 1

Server Basics and Administering Global Settings

Chapter 1

Getting Started with Sun ONE Application Server Administration

This chapter describes the basics of administering Sun ONE Application Server: where to find general information about the server, how to access the server's user interfaces, and how to access online help.

This chapter includes the following sections:

- About Sun ONE Application Server
- Configuring the Bundled Solaris Version
- Using the Administration Interface
- Using the Command-line Interface
- Accessing the Administration Server
- Accessing Application Server Instances
- Using Sun ONE Studio
- About Configuration Files
- Using the License Commands

About Sun ONE Application Server

The Sun ONE Application Server provides a robust J2EE platform for the development, deployment, and management of e-commerce application services to a broad range of servers, clients, and devices. Key features include transaction management, performance, scalability, security, and enterprise application integration.

Sun ONE Application Server supports services from web publishing to enterprise-scale transaction processing, while enabling developers to build applications based on JavaServer Pages (JSP™), Java Servlet and Enterprise JavaBeans™ (EJB™) technology.

It contains the following basic tools for administrators:

- Multiple administrative domains that allow different administrators to create and manage their own set of application server instances.
- An Administration Server that provides administration facilities (one Administration Server per domain)
- A graphical user interface for server administration (the Administration interface)
- A command-line interface you can use to perform the same tasks as the Administration interface

Use these tools to perform all administrative functions, including:

- Administering domains
- Managing server instances
- Deploying applications
- Monitoring the server
- Using log files
- Administering resources
- Administering a Message Queue server
- Using transaction services
- Using Corba/IIOP clients
- Configuring the web server plug-in
- Configuring J2EE containers
- Administering HTTP server features

For more information about the Sun ONE Application Server architecture and features, and for initial steps to take with your Sun ONE Application Server, see the *Sun ONE Application Server Getting Started Guide*.

Configuring the Bundled Solaris Version

This guide refers to two kinds of Sun ONE Application Server installations for Solaris: Solaris 9 bundled and unbundled. If you received your copy of the Sun ONE Application Server as part of the Solaris 9 installation, you have the Solaris bundled version. If you have a standalone copy of the Sun ONE Application Server, you have the unbundled version.

NOTE If you are using the unbundled Solaris version of the Sun ONE Application Server, or if you are using the Windows version, please skip this section and move on to "Using the Administration Interface" on page 35.

If you are using the version of the Sun ONE Application Server bundled with Solaris 9, you need to perform some extra configuration steps before you can start using the server.

When you install an unbundled version of the Sun ONE Application Server, as part of the installation process a domain, an Administration Server, and a server instance are automatically created.

When you use the Solaris 9 bundled version, you have to create these items manually, as well as perform other steps. Once you have performed these initial steps, you can take advantage of all Sun ONE Application server features, including adding additional administrative domains and server instances.

The following topics are discussed in this section:

- Creating an Administrative Domain
- Starting the Administration Server
- Creating an Application Server Instance
- Deploying Applications

Creating an Administrative Domain

Multiple administrative domains allow different administrative users to create and manage their own domains. A domain is a set of instances, created using a common set of installed binaries in a single system. Each domain has one Administration Server.

When you create a new domain, you specify:

- A port number for the Administration Server. The default when you install an unbundled version is 4848.
- An administration username and password. These passwords are required when you access the Administration Server, either when you access the Administration interface or when you run the command-line interface.
- The domain location.

You must create a domain using the command-line interface's `asadmin` utility's `create-domain` command. For more information on creating an administrative domain, see Chapter 3, "Configuring Administrative Domains."

Starting the Administration Server

When you create an administrative domain, you create an Administration Server. The Administration Server is a special instance of the Sun ONE Application Server that serves the Administration interface and provides administrative facilities for the command-line interface.

In order to use the Administration interface or to use many of the commands in the command-line interface, you must have a running Administration Server. For information on starting the Administration Server, see "Starting the Administration Server" on page 47.

Creating an Application Server Instance

Once you have created a domain and started the Administration Server, you need to create an application server instance. Each application server instance has its own J2EE configuration, J2EE resources, application deployment areas, and server configuration settings.

You can create an application server instance through the Administration interface or through the command-line interface. The server instance is created in a folder within the domain.

On the unbundled version, the server instance created at installation is called `server1`. You will often see `server1` used in examples throughout the documentation.

For more information about creating an application server instance, see "Adding an Application Server Instance" on page 78.

Deploying Applications

Once you have created a domain, started the Administration Server, and added an application server instance, you can deploy applications to that instance. For more information, see Chapter 13, “Deploying Applications”.

Using the Administration Interface

Use the Administration interface to configure all aspects of your server. This section contains the following topics:

- Accessing the Administration Interface
- Using Tabs
- Using Buttons
- Accessing Online Help
- Exiting the Administration Interface

NOTE Some aspects of server configuration and corresponding Administrative interfaces are still undergoing change. An unstable interface may be removed and replaced with a cleaner and more stable version in a subsequent product release. Most server configuration and administrative interfaces will remain the same or change in a compatible manner; however some aspects may change incompatibly. Product documentation for future releases will clearly describe incompatible when they do occur.

Accessing the Administration Interface

The Administration interface for Sun ONE Application Server runs on an HTTP server called the Administration Server. The Administration Server is installed when you install Sun ONE Application Server if you are using an unbundled version. You have to create an administrative domain and Administration Server if you are installing the bundled version. For more information, see “Configuring the Bundled Solaris Version” on page 33.

The Administration Server must be running in order for you to use the Administration interface. For information on starting the Administration Server, see “Starting the Administration Server” on page 47.

When you installed the Sun ONE Application Server or created a domain, you chose a port number for the Administration Server, or used the default port of 4848. To access the Administration interface, in a web browser type:

http://*hostname.domain*:*port*/

For example:

http://austen.sun.com:4848/

If you are accessing the Administration Server from the same machine you installed Sun ONE Application Server on, you can use:

http://*localhost*:4848

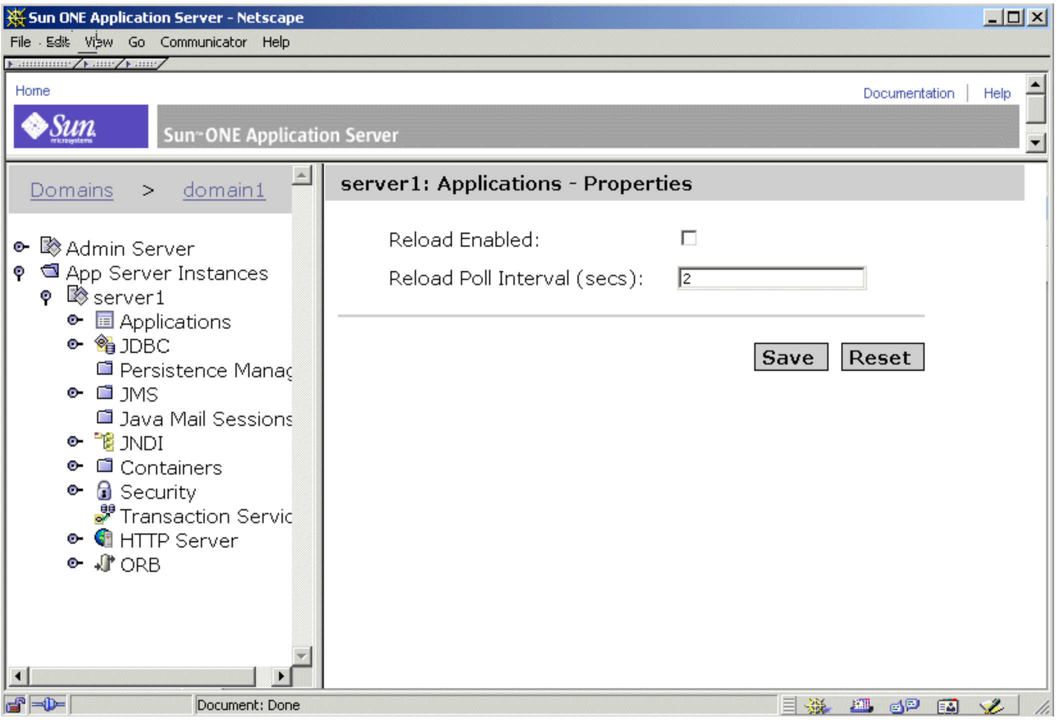
You are prompted for a user name and password, which the installing user set during installation or when you created the domain and Administration Server.

You can access the Administration interface from a remote location as long as you have access to a browser. You can access it from any machine that can reach the server over the network.

On Windows, you can access the Administration interface from the Start menu by choosing Programs, then Sun Microsystems, then Sun ONE Application Server 7, then Start Admin Console.

The following figure shows the Administration interface.

Figure 1-1 The Sun ONE Application Server Administration Interface



server1: Applications - Properties

Reload Enabled: □

Reload Poll Interval (secs): 2

The left pane is a tree view of all items you can configure in the Sun ONE Application Server. To use the Administration interface, click an item in the left pane. The right pane displays the page associated with that item.

If an item in the left pane has an open/close symbol next to it, you can expand that item to subitems by clicking on the open/close symbol. When the tree item is not expanded, the symbol looks like this:

Figure 1-2 Closed Symbol

When the tree item is expanded, the symbol looks like this:

Figure 1-3 Open Symbol

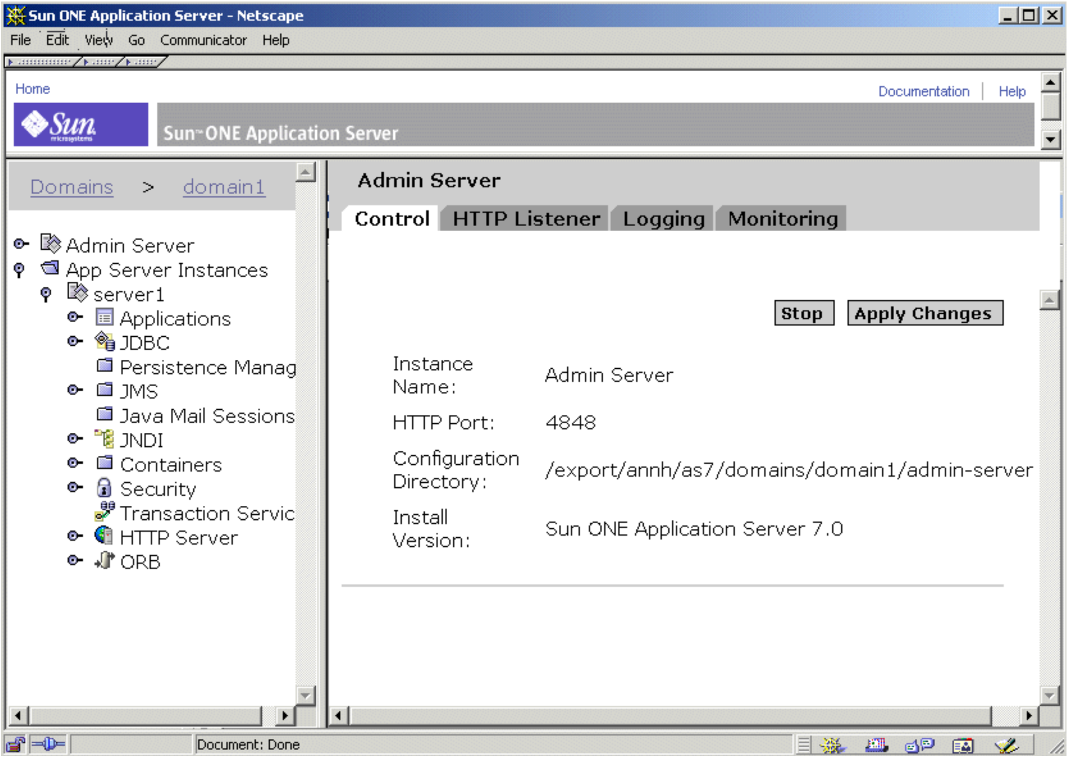
Using Tabs

Some portions of the Administration interface contain tabs which you use to navigate to other pages. These tabs appear in a separate pane at the top of the right pane.

To use the tabs, click the tab name. Some tabs take you directly to a page, while others have a list of pages available from the tab which appear below the tab names. Click a page name to go to that page.

The following figure shows the Administration interface with tabs:

Figure 1-4 Administration Interface With Tabs



Instance Name: Admin Server

HTTP Port: 4848

Configuration Directory: /export/annh/as7/domains/domain1/admin-server

Install Version: Sun ONE Application Server 7.0

Using Buttons

The following standard buttons are available in the Administration interface.

Table 1-1 Standard Administration interface Buttons

Button	Action Performed
Cancel	Cancels without saving your changes and returns to previous page.
Delete	Deletes an item. Often you click Select next to an item to indicate what to delete when you click Delete.
New	Displays a page for creating a new item. For example, clicking New on the Application Server Instances page displays the Create New Instance page.
OK	Saves your entries. If you've made configuration changes to a Sun ONE Application Server instance, you must apply your changes in order for them to take effect. For more information, see "Applying Changes to an Application Server Instance" on page 79.
Reset	Resets the values on the page to the default values.
Save	Saves your entries. If you've made configuration changes to a Sun ONE Application Server instance, you must apply your changes in order for them to take effect. For more information, see "Applying Changes to an Application Server Instance" on page 79.

Other buttons are available depending upon the needs of a particular screen.

Accessing Online Help

You can access help for any page in the Administration interface by clicking the Help button in the banner at the top of the Administration interface. The online help describes the use of the page you are accessing and gives information about what to enter in the fields on the page.

Figure 1-5 Online Help

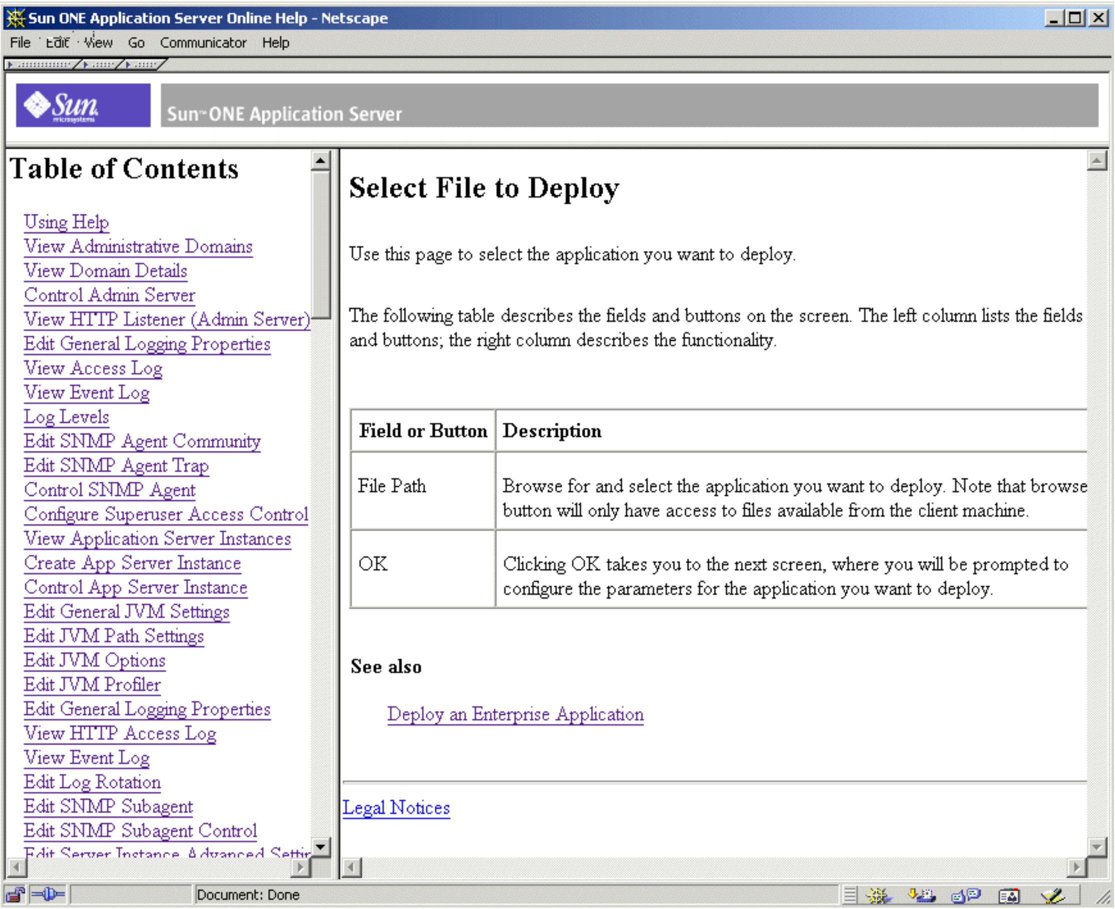

Sun ONE Application Server Online Help - Netscape
File Edit View Go Communicator Help
Sun
Sun™ONE Application Server
Document: Done

Table of Contents

- Using Help
- View Administrative Domains
- View Domain Details
- Control Admin Server
- View HTTP Listener (Admin Server)
- Edit General Logging Properties
- View Access Log
- View Event Log
- Log Levels
- Edit SNMP Agent Community
- Edit SNMP Agent Trap
- Control SNMP Agent
- Configure Superuser Access Control
- View Application Server Instances
- Create App Server Instance
- Control App Server Instance
- Edit General JVM Settings
- Edit JVM Path Settings
- Edit JVM Options
- Edit JVM Profiler
- Edit General Logging Properties
- View HTTP Access Log
- View Event Log
- Edit Log Rotation
- Edit SNMP Subagent
- Edit SNMP Subagent Control
- Edit Server Instance Advanced Settir

Select File to Deploy

Use this page to select the application you want to deploy.

The following table describes the fields and buttons on the screen. The left column lists the fields and buttons; the right column describes the functionality.

Field or Button	Description
File Path	Browse for and select the application you want to deploy. Note that browse button will only have access to files available from the client machine.
OK	Clicking OK takes you to the next screen, where you will be prompted to configure the parameters for the application you want to deploy.

See also

Deploy an Enterprise Application

Legal Notices

You can navigate to help for other pages using the table of contents in the left pane of the help window. For help on using help, see the first topic, Using Help, in the online help table of contents.

Exiting the Administration Interface

There is no explicit exit or logout button in the Administration interface. To exit, close the browser you are using to access the Administration interface. Also, close other instances of the same browser that may be running on your machine.

Using the Command-line Interface

Sun ONE Application Server contains a command-line interface. You can use the `asadmin` utility and the commands associated with it to perform the same set of tasks as you can perform in the Administration interface. For example, you can start and stop application server instances, configure the server, and deploy applications.

You can use these commands either from a command prompt in the shell, or you can call them from other scripts and programs. You can use these commands to automate repetitive administration tasks.

For more information about using the command-line interface, and for a list of commands, see Appendix A, “Using the Command Line Interface.”

Accessing the Administration Server

The Administration Server is a special instance of the Sun ONE Application Server that serves the Administration interface and provides administrative facilities for the Administration interface and the command-line interface. It manages the configuration, deployment, and monitoring facilities for these interfaces and tools, and so must be running for them to work.

To configure Administration Server properties, access the Administration interface. Click Admin Server in the left pane to display configuration settings for the Administration Server.

For more information on the Administration Server, see Chapter 2, “Setting Administration Server Preferences.”

Accessing Application Server Instances

You can have multiple Sun ONE Application Server instances. Each application server instance has its own configuration, resources, and application deployment areas that are independent of other application server instances. If you change the configuration for one application server instance, you do not change the configurations of other application server instances. The Administration interface contains an item for every application server instance you create. One application server instance is created at installation time if you are using the unbundled version of the software. You can create others if you like.

If you are using Solaris 9 bundled version, the application server instance is not automatically created. For more information, see "Configuring the Bundled Solaris Version" on page 33.

For more information on application server instances, see Chapter 4, "Using Application Server Instances."

Using Sun ONE Studio

To deploy applications, in addition to using the Administration interface and the command-line interface, you can also use Sun ONE Studio 4. For more information about Sun ONE Studio, see the *Sun ONE Studio 4, Enterprise Edition for Java with Application Server 7 Tutorial* at http://docs.sun.com

About Configuration Files

When you make a change to the server settings through the Administration interface or command-line interface, the Administration Server changes the underlying configuration files. These files contain setting for the Administration Server and for individual application server instances.

For more information on the files and what settings they contain, see the *Sun ONE Application Server Administrator's Configuration File Reference.*

Using the License Commands

When you purchase Sun ONE Application Server, a license of the appropriate type is installed automatically during installation. For more information on the types of licenses and their restrictions, see the *Sun ONE Application Server Installation Guide*.

Sun ONE Application Server contains command-line utilities for managing your licenses after installation.

To upgrade your license after installation, you can use the `asadmin` utility's `install-license` command for Windows machines. This command must be run locally on the machine on which you have installed the Sun ONE Application Server. You must stop any running application server before installing a license.

It has the following syntax:

```
asadmin install-license
```

For unbundled Solaris package-based installations, the following syntax is used for license installation:

`pkgadd -d` *full_path* SUNWaslco

For example,

pkgadd -d /*install_dir*/ pkg SUNWaslco

For Solaris 9 bundled license installation, the following syntax is used for license installation:

`pkgadd -d` *full_path* SUNWaslc

To get information on your license, use the `display-license` command, which has the following syntax:

`asadmin display-license [--user` *admin_user*`] [--password` *admin_password*`]`
`[--passwordfile` *password_file*`][--host` *localhost*`] [--port` *admin_port*`]`
`[--local=`*true/false*`]`

This command can be run locally or remotely, depending on the value of the local option. For example, to run the command from the local machine, taking the defaults for the host and the port number, the syntax is:

```
asadmin display-license --local
```

The output describes the type of license currently installed (for example, evaluation) the expiration date, if it has one, the number of instances per Administration Sever your license allows, and whether remote administration is allowed or not.

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Chapter 2

Setting Administration Server Preferences

The Administration Server is a special instance of the Sun ONE Application Server that serves the Administration interface and provides administrative facilities for the Administration interface and the command-line interface. It manages the configuration, deployment, and monitoring facilities for the Sun ONE Application Server. This chapter explains how to configure the Administration Server.

This chapter includes the following topics:

- About the Administration Server
- Starting the Administration Server
- Shutting Down the Administration Server
- Accessing the Administration Server Settings
- Viewing Administration Server Control Settings
- Applying Changes to the Administration Server
- Editing HTTP Listener Settings for the Administration Server
- Setting SNMP, Logging, and Security Preferences

About the Administration Server

The Administration Server is a special instance of the Sun ONE Application Server that provides administrative facilities for the Administration interface and the command-line interface. It manages the configuration, deployment, and monitoring facilities for these interfaces. It serves the Administration interface pages. It must be running in order for you to use the Administration interface and to run most commands in the command-line interface.

An Administration Server is installed when you install the Sun ONE Application Server (unbundled version) or create a new domain. You can only have one Administration Server per domain, though you may create multiple Sun ONE Application Server instances that are managed by an Administration Server. You can access configuration settings, deployed applications, and other server features for each application server instance by using the Administration Server.

For more information on administrative domains, see Chapter 3, "Configuring Administrative Domains."

If you are using the unbundled Sun ONE Application Server, when you installed the Sun ONE Application Server you chose a port number for the Administration Server, or used the default port of 4848.

If you are using the Solaris 9 bundled Sun ONE Application Server, you must create a domain and an Administration Server manually. For more information on configuring the Solaris 9 bundled version, see "Configuring the Bundled Solaris Version" on page 33.

To access the Administration interface, in a web browser type:

http://*hostname.domain*:*port*/

Note that *domain* here is not your Sun ONE Application Server administrative domain, but your domain name.

For example:

http://austen.sun.com:4848/

If you are accessing the Administration Server from the same machine you installed Sun ONE Application Server on, you can use:

http://localhost:4848

You are prompted for a user name and password.

Starting the Administration Server

To start or restart the Administration Server, use one of the methods described in the following topics:

- Using the startserv Script
- Using the Command-Line Interface
- Using the Services Window (Windows)
- Using the Start Menu (Windows)

Using the startserv Script

To start the Administration Server using the start script, log in as root if the server runs on ports with numbers lower than 1024 (UNIX); otherwise, log in as root or with the server's user account. At the command-line prompt go to the directory:

install_dir / domains / *domain_dir* / admin-server / bin

where *install_dir* is the directory where you installed the server, and *domain_dir* is the administrative domain directory.

For UNIX, type:

```
./startserv
```

You can use the optional parameter `-i` at the end of the line. The server normally runs as a background process. The `-i` option prevents the server from putting itself in the background. This option is useful if you are running the server using `/etc/inittab`.

For Windows, run the `startserv.bat` file.

NOTE If the server is already running, the `startserv` command will fail. You must stop the server first, then use the `startserv` command. Also, if the server startup fails, you should kill the process before trying to restart it.

Using the Command-Line Interface

The command-line interface's `asadmin` utility has a command `start-domain` that you can use to start your Application Server and all associated Sun ONE Application Server instances. You can only run this command locally, that is, from the machine where your Sun ONE Application Server is installed. This command requires no arguments.

You can also use the command `start-domain` to start all instances within an administrative domain. It uses the syntax:

```
asadmin start-domain [--domain domain-name]
```

For more information on using the command-line interface, see the online help for the command-line interface and Appendix A, "Using the Command Line Interface."

Using the Services Window (Windows)

To start the server by using the Services Control Panel in Windows, follow these steps:

1. In the Control Panel click Administrative Tools.
2. Click Services.
3. Scroll through the list of services and double-click the Application Server 7.0 Administration Server service.

   Select the Administration Server for the given domain. The Name column in the Services window displays the name of an Administration Server as Sun App Server Admin Server (*your_domain_name*:admin-server).
4. Click Start.
5. Click OK.

The Application Server 7.0 Administration Server service starts automatically when you start your computer.

Using the Start Menu (Windows)

To start the server by using the Windows Start menu, follow these steps:

1. From the Start menu, choose Programs.

2. Choose Sun Microsystems.
3. Choose Sun ONE Application Server 7
4. Click Start Application Server.

Shutting Down the Administration Server

Once the Administration Server is started it runs constantly, listening for and accepting requests. You might want to stop and restart your server if, for instance, you change the Administration Server logging preferences or the port that the Administration Server's HTTP Listener listens on.

When you stop the Administration Server, it stops accepting new connections. Then it waits for all outstanding connections to complete. While the Administration Server is stopped, you cannot use the Administration interface or command-line interface.

You can stop the server using one the methods described in the following topics:

- Shutting Down Using the Administration Interface
- Shutting Down Using the stopserv Script
- Shutting Down Using the Command-Line Interface
- Shutting Down Using the Services Window (Windows)

After you shut down the server, it may take a few seconds for the server to complete its shut-down process.

Shutting Down Using the Administration Interface

To shut down the Administration Server using the Administration interface, follow these steps:

1. In the left pane, click Admin Server.
2. Click the Control tab.
3. Click Stop.

   The Administration Server shuts down immediately when you click this link. There is no additional screen.

Shutting Down Using the stopserv Script

To stop the Administration Server manually, at the command prompt go to the directory:

install_dir / domains / *domain_dir* / admin-server / bin

where *install_dir* is the directory where you installed the server and *domain_dir* is the domain directory.

For UNIX, type:

```
./stopserv
```

You must run this command as the user that is running the server.

If you used the `*/etc*/inittab` file to restart the server you must remove the line starting the server from `*/etc*/inittab` and type `kill -1` before you try to stop the server. Otherwise, the server restarts automatically after it is stopped.

For Windows, run the `stopserv.bat` file.

Shutting Down Using the Command-Line Interface

You can stop the Administration Server using the command-line interface `asadmin` utility's `shutdown` command. This command requires no arguments and can be run either locally or remotely.

You can also stop the Administration Server and all the associated Sun ONE Application Server instances using the command-line interface `asadmin` utility's `stop-appserv` command. You can only run this command locally, that is, from the machine where your Sun ONE Application Server is installed. This command requires no arguments.

You can also shut down the Administration Server by shutting down the domain using the `stop-domain` command. This command shuts down all instances in the domain, including the Administration Server, by default. You can also configure it to shut down all instances in the domain *except* the Administration Server. This command has the following syntax:

```
asadmin stop-domain [--user admin_user] [--password admin_password]
[--host admin_host] [--port admin_port] [--local=true/false] [--domain
domain_name] [--adminserv=true/false] [--passwordfile file_name] [--secure |
-s]
```

If you use the `local` option, the command runs locally. If you use the `--adminserv=false` option, the command will not stop the Administration Server. However, `--adminserv` is set to `true` by default, so the Administration Server is stopped by default.

For more information on using the command-line interface, see the online help for the command-line interface and Appendix A, "Using the Command Line Interface."

Shutting Down Using the Services Window (Windows)

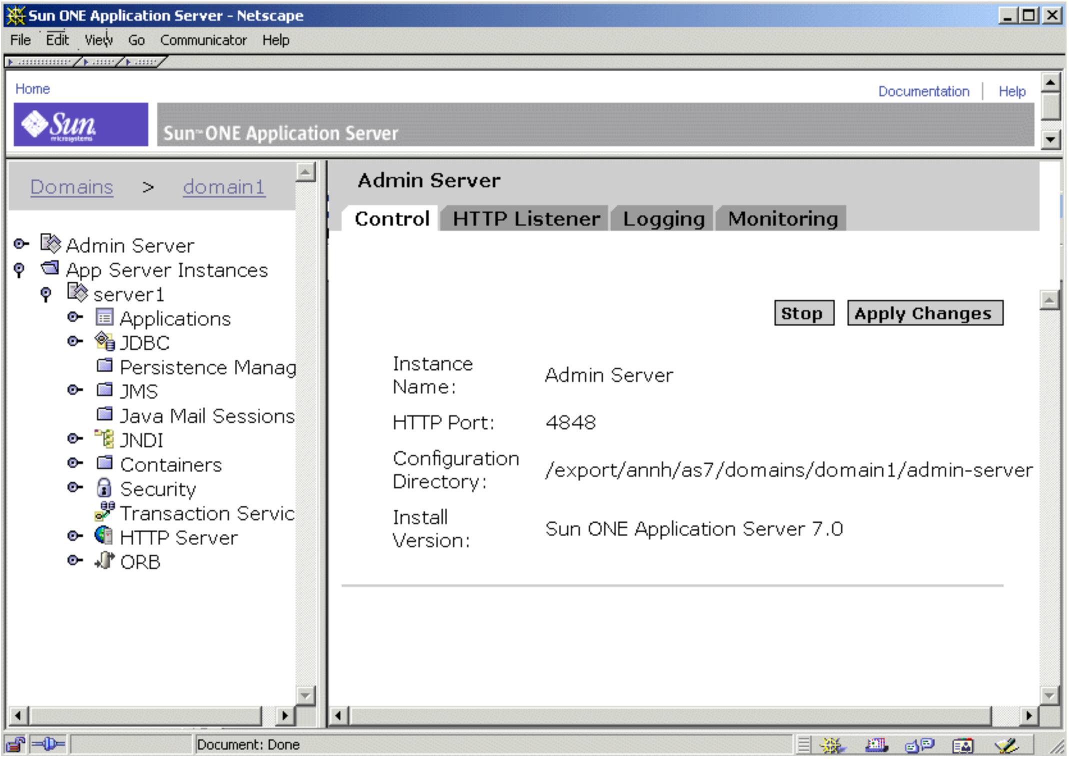
To shut down the Administration Server using the Services window, follow these steps:

1. In the Control Panel click Administrative Tools.
2. Click Services.
3. Scroll through the list of services and double-click the Sun Application Server 7 Admin Server service.
4. Click Stop.
5. Click OK.

Accessing the Administration Server Settings

To access the Administration Server settings, in the left pane of the Administration interface click Admin Server. The settings for the Administration Server appear in the right pane, organized by a set of tabs.

Figure 2-1 Administration Server User Interface



Instance Name: Admin Server

HTTP Port: 4848

Configuration Directory: /export/annh/as7/domains/domain1/admin-server

Install Version: Sun ONE Application Server 7.0

Click a tab to access the settings for a particular functional area. Sometimes clicking a tab takes you directly to a page, and sometimes clicking a tab gives you a list of pages to choose from.

This chapter covers the Control and HTTP Listener tabs. For information on Monitoring and SNMP settings, see Chapter 6, “Monitoring the Sun ONE Application Server.” For information on logging, see Chapter 5, “Using Logging.”

Viewing Administration Server Control Settings

The Administration Server control settings show the instance name (Admin Server), what port your Administration Server runs on, what directory contains the configuration files, and what version of the Sun ONE Application Server software you are running.

To view these settings:

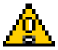
1. In the left pane, click Admin Server.
2. Click the Control tab.

Applying Changes to the Administration Server

When you change the Administration Server's configuration information using the Administration interface or the command line interface you may need to apply your changes in order for your changes to take effect. This is also called reconfiguring the server. When you apply your changes, all changes made to the configuration since the last time you applied changes take effect.

If you've made changes to the Administration Server's configuration that require you to apply changes, a yellow icon appears next to the Admin Server in the left pane's tree view.

Figure 2-2 Warning Icon



To apply configuration changes:

1. In the left pane, click Admin Server.
2. Click the Control tab.
3. Click Apply Changes.

   When the changes are applied, the screen displays a message.

Editing HTTP Listener Settings for the Administration Server

Before the server can process a request, it must accept the request via an HTTP listener.

With the unbundled version of the Sun ONE Application Server, an HTTP listener for the Administration Server, `http-listener-1`, is created automatically when you install. This HTTP listener uses the IP address 0.0.0.0 and the port number you specified as your Administration Server port number during installation. The default is 4848. You cannot delete the default HTTP listener.

For an administrative server instance (in a domain), only the HTTP listener has the `http-listener-1` id. Another words, if you create an administration server instance, only one HTTP listener can act in either HTTP or HTTPS protocol at any point in time. (It also means that you cannot have both HTTP and HTTPS connections to an administration server simultaneously.) For more information about configuring the Solaris 9 bundled version, see "Configuring the Bundled Solaris Version" on page 33.

The HTTP listener is where you activate and configure SSL/TLS security settings for your Administration Server.

In addition, you specify the number of acceptor threads (sometimes called accept threads) in the HTTP listener. Accept threads are threads that wait for connections. The threads accept connections and put them in a queue where they are then picked up by worker threads. Ideally, you want to have enough accept threads so that there is always one available when a new request comes in, but few enough so that they do not provide too much of a burden on the system. The default is 1. A good rule is to have one accept thread per CPU on your system. You can adjust this value if you find performance suffering. For more information on performance, see the *Sun ONE Application Server Performance Tuning and Sizing Guide*.

To edit your Administration Server's HTTP listener settings:

1. In the left pane, click Admin Server.
2. Click the HTTP Listeners tab.
3. Make the desired changes and click OK.

For more information on HTTP listeners, see the online help.

Setting SNMP, Logging, and Security Preferences

For information on SNMP settings, see Chapter 6, “Monitoring the Sun ONE Application Server.” For information on logging, see Chapter 5, “Using Logging.” For information on security settings, see the *Sun ONE Application Server Administrator’s Guide to Security*.

Chapter 3

Configuring Administrative Domains

This chapter explains how to set up and administer administrative domains with your Sun ONE Application Server.

This chapter includes the following topics:

- About Administrative Domains
- Configuring Domains
- Recreating the Domain Registry

About Administrative Domains

Administrative domains provide a basic security structure whereby different administrators can administer specific groups (domains) of application server instances on a machine. By dividing the application server instances up like this it is possible to share a single machine between disparate organizations, each with their own administrator.

Within the Sun One Application Server every application server instance is a member of one domain. It is not required that there be any more than one domain, however, multiple domains are supported as a useful feature if so desired.

Administrative security is established for local commands using the underlying operating system's security mechanisms (that is, via file permissions). Remote command security is established using a username/password pair to communicate with a specific admin server. Administrative domains don't utilize any other security constructs.

This section describes the following topics:

- Implementing Administrative Domains

- Directory Structure
- Process / Port Structure

Implementing Administrative Domains

A domain is implemented using files, operating system processes and ports. Each domain has a unique name.

Directory Structure

There are files (configuration, executables, and so on) that are shared by all domains within an installation. What is important for this discussion are those files which are specific to a domain.

The files specific to a domain all share a common root directory known as the domain directory, and whose name is the name of the domain. Under the domain directory is one directory per instance, each named after the instance, and under each of those instance directories are instance-specific files.

A domain directory can be constructed anywhere in the file system (in accordance with security permissions and other operating system level constraints). Unless a user chooses otherwise, domain directories are constructed under a default directory (known as the domains directory). However a user can choose to create a domain directory anywhere.

Process/Port Structure

When a domain is running it consumes operating system processes and ports. Specifically, for each instance running within a domain (including the domain's admin server) there is a process and a port.

Configuring Domains

Domains can be created, deleted, listed, started and stopped using commands specifically designed for that purpose.

Creation, deletion and starting of domains can only be done locally, whereas listing and stopping can be done both locally and remotely.

The deletion, starting and stopping commands all take a domain name. This name is optional if there is only one domain. The command will give an error if no domain was given but there are multiple domains configured.

This section contains the following topics:

- Creating Domains
- Deleting Domains
- Listing Domains
- Starting Domains
- Stopping Domains

Creating Domains

Domains are created using the `create-domain` command. This command is local only.

Synopsis:

```
asadmin create-domain [--path domain_path] [--sysuser  sys_user]
[--passwordfile file_name] --adminport  port_number --adminuser admin_user
--adminpassword password domain_name
```

Example: creating a domain in the default location

```
$ asadmin create-domain --adminport 123 --adminuser MyAdmin
--adminpassword MyPassword MyDomain
```

This example creates a domain called MyDomain in the default location (that is, the domains directory). The administration server will listen on port 123, the admin user name will be MyAdmin and the password will be MyPassword. The domain directory and files underneath it will be owned by the operating system user who executed this command. In addition, the operating system processes will run as the user who executed this command.

If there's already a domain called MyDomain then an error message is returned.

(Note that instead of using the password on the command line, which could be a security issue, you can put the password in a file and pass it through in using the `--passwordfile` option).

Example: creating a domain somewhere other than the default location

```
$ asadmin create-domain --path $HOME --adminport 123 --adminuser
MyAdmin --adminpassword MyPassword MyDomain
```

This example is similar to the first, except that the domain directory will now be located underneath the user's `$HOME` directory rather than under the default domains directory.

Example: creating a domain for another user (UNIX only)

```
# asadmin  create-domain --user AnotherUser --adminport 123
--adminuser MyAdmin --adminpassword MyPassword MyDomain
```

This example is similar to the first, except that the domain and its files will be owned by the user AnotherUser, as will the operating system processes

Using the `--sysuser` option provides the ability for one user to construct domains that other users can subsequently administer. This option requires that the user running the create-domain command be root.

User Permissions on UNIX Platforms

In order for a non-root user to create and delete administrative domains, you must add the user ID to a UNIX group that has write permissions to the domain configuration files:

1. Create a UNIX group that will be the group which is applied to the installation-wide domain configuration files. For example, a UNIX group named asadmin.

2. Set the installation-wide domain configuration files located under /etc/appserver to be owned by the newly created UNIX group.

   The files are named `domains.bin` and `domains.lck`. For example, after changing the group assigned to these files:

   ```
   -rw-r--r-- 1 root asadmin 0 Sep 18 14:34 domains.bin
   ```

   ```
   -rw-r--r-- 1 root asadmin 0 Sep 18 14:34 domains.lck
   ```

3. Enable write access to these files by the newly created UNIX group. In this example, the resulting permissions would look like the following:

   ```
   -rw-rw-r-- 1 root asadmin 0 Sep 18 14:34 domains.bin
   ```

   ```
   -rw-rw-r-- 1 root asadmin 0 Sep 18 14:34 domains.lck
   ```

4. Add the user ID to the UNIX group.

Alternatively, if you do not want to provide non-root users with write access to the installation-wide configuration files, you can create an administrative domain on behalf of a user. During creation of a new administrative domain, specify the `--sysuser` and `--path` options to identify the UNIX user ID that will own the domain's directories and files and the location under which the administrative domain will be created. For an example, see "Example: creating a domain for another user (UNIX only)" on page 60.

Once an administrative domain is created under a user ID, the user may create new application server instances and perform a wide variety of administrative operations on the application server instances. The user ID does not need to belong to the UNIX group that has write privileges for the administrative domain configuration file. Membership in the UNIX group is required only to create and delete administrative domains.

Deleting Domains

Domains are deleted using the `delete-domain` command. Only the operating system user (or root) who can administer the domain can execute this command successfully. This command is local only.

Synopsis:

```
asadmin delete-domain [domain_name]
```

Example: deleting a domain

```
$ asadmin delete-domain MyDomain
```

This example deletes the domain called MyDomain on the local machine.

Listing Domains

The domains created on a machine can be found using the `list-domains` command.

This command can operate both locally and remotely.

Synopsis:

```
asadmin list-domains [--host host] [--port port] [--password password]
[--user user]
```

Example: listing the domains on a local machine

```
$ asadmin list-domains
```

```
domain1    [/opt/ias/build/domains/domain1]
```

Example: listing the domains on the local machine using the remote option:

```
$ asadmin list-domains --user admin --password password --host
localhost --port 4848
```

```
domain1    [/opt/ias/build/domains/domain1]
```

Starting Domains

Domains can be started using the `start-domain` command. This starts up the domain's admin server and all other instances in the domain.

This command can only be run locally.

Synopsis:

```
asadmin start-domain [--domain domain_name]
```

Example: starting the only domain on a machine:

```
$ start-domain
```

```
Instance domain1:admin-server started

Instance domain1:server1 started

Domain domain1 Started.
```

Stopping Domains

Domains can be stopped using the `stop-domain` command. The user can choose to stop every instance within a domain, or all the instances except the admin server, thus leaving the domain able to be remotely administered.

This command can be run both locally and remotely.

Synopsis:

```
asadmin stop-domain [--user admin_user] [--password admin_password]
[--host host_name] [--port port_name] [-- local=false] [--domain
domain_name] [--adminserv=true] [--passwordfile file_name] [--secure |
-s]
```

Example: Stopping all instances in a domain except for the admin server instance.

```
$ asadmin asadmin stop-domain --user admin --password password
--host localhost --port 4848 --adminserv=false --domain domain1

DomainStoppedRemotely
```

Recreating the Domain Registry

For implementation purposes the details of each domain (its name, location, ports used, and so on) are recorded in a file known as the domain registry.

Under normal operating conditions you should not have to do anything with the domain registry directly, since any modification or use of the domain registry is entirely encapsulated by the commands used to administer the system. However, because the domain registry is a file, it can become corrupted (for example, when a script goes wrong, or when someone inadvertently deletes the registry, and so on), and in those cases you may have to recreate the file.

NOTE You can access the domain registry through the command line interface by using the `asadmin` command.

If the registry becomes corrupted, perform the following procedure to recreate the registry:

1. Get a list of all the domains, and the directory (default or non-default) that they’re located in.
2. Rename each directory (for example, append each directory name with the suffix “.bak”)
3. Create each domain again in its original location, using default values for ports, passwords, and so on.
4. Delete each new domain directory and replace it with the original directory.

5. For each domain execute the `reconfig` command. This will cause the domain registry to be updated with the values from the old domain.

Part 2

Managing an Individual Server Instance

Chapter 4, “Using Application Server Instances”

Chapter 5, “Using Logging”

Chapter 6, “Monitoring the Sun ONE Application Server”

Chapter 7, “Configuring the Web Server Plugin”

Chapter 8, “Configuring J2EE Containers”

Chapter 9, “Using Transaction Services”

Chapter 10, “Configuring Naming and Resources”

Chapter 11, “Using the JMS Service”

Chapter 12, “Configuring the Server For CORBA / IIOP Clients”

Chapter 13, “Deploying Applications”

Chapter 4

Using Application Server Instances

This chapter describes how to create, delete, configure, start, and stop Sun ONE Application Server instances.

This chapter includes the following topics:

- About Application Server Instances
- Starting and Stopping an Application Server Instance
- Starting the Application Server Instance in Debug Mode
- Setting the Termination Timeout
- Restarting an Application Server Instance Automatically (UNIX)
- Restarting an Application Server Instance Manually (UNIX)
- About the Watchdog
- Adding an Application Server Instance
- Deleting an Application Server Instance
- Applying Changes to an Application Server Instance
- Viewing Application Server Instance Status
- Configuring JVM Settings
- Configuring Logging Setting and Monitoring Settings
- Changing Application Server Instance Advanced Settings

About Application Server Instances

Sun ONE Application Server creates one application server instance, called `server1`, when you install the unbundled version of the software. You can delete the `server1` instance and create a new instance with a different name if you prefer.

If you are using the Solaris 9 bundled version of the software, you must create your own server instance. For more information, see "Configuring the Bundled Solaris Version" on page 33.

Each application server instance has its own J2EE configuration, J2EE resources, application deployment areas, and server configuration settings. Changes to one application server instance have no effect on other application server instances. You can have many application server instances within one administrative domain. Within a domain, all the server instances have the same Administration Server. For more information on domains, see Chapter 3, "Configuring Administrative Domains."

For many users, one application server instance meets their needs. However, depending upon your environment, you might want to create one or more additional application server instances. For example, in a development environment you can use different application server instances to test different Sun ONE Application Server configurations, or to compare and test different application deployments. Because you can easily add or delete an application server instance, you can use them to create temporary "sandbox" areas to experiment with while developing.

In addition, for each application server instance you can also create virtual servers. Within a single installed application server instance you can offer companies or individuals domain names, IP Addresses, and some administration capabilities. For the users, it is almost as if they have their own web server, without the hardware and basic server maintenance. These virtual servers do not span application server instances. For more information about virtual servers, see Chapter 15, "Using Virtual Servers."

In operational deployments, for many purposes you can use virtual servers instead of multiple application server instances. However, if virtual servers don't meet your needs, you can also use multiple application server instances.

Starting and Stopping an Application Server Instance

A Sun ONE Application Server instance is not started automatically. Once you start an instance, it the instance runs until you stop it. When you stop an application server instance, it stops accepting new connections, then waits for all outstanding connections to complete. If your machine crashes or is taken offline, the server quits and any requests it was servicing may be lost.

You can start and stop the application server instance using one of several methods, covered in the following topics:

- Using the Start and Stop Buttons in the Administration Interface
- Using the start-instance and stop-instance Commands
- Using the Windows Services (Windows)
- Using the startserv and stopserv Scripts

NOTE If you have a security module installed with your server, you will be required to enter the appropriate passwords before starting or stopping the server.

If you have a server certificate installed, the Sun ONE Application Server prompts the administrator for the key database password before starting up. If you want to be able to restart an unattended Sun ONE Application Server, you need to save the password in a `password.conf` file. Only do this if your system is adequately protected so that this file and the key databases are not compromised. For more information on creating and using `password.conf`, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

NOTE On UNIX, some Sun ONE Application Server installations may require access to more memory and/or file descriptors than your operating system allows by default. If you are unable to start the server, check the resource limits imposed by your operating system using the `ulimit` command. Your operating system's `ulimit` man page should provide more information.

Using the Start and Stop Buttons in the Administration Interface

To start and stop the server using the Administration interface:

1. In the left pane, under App Server Instances, click the name of the instance to start or stop.
2. In the right pane, click Start or Stop; or in the General Tab, click Start or Stop.
3. You see a message when the application server instance is started or stopped successfully.

Using the start-instance and stop-instance Commands

Using the command-line interface utility `asadmin`, you can start and stop your application server instances either from the command prompt or from a script. Use the commands `start-instance` and `stop-instance`.

These commands have the following syntax:

```
start-instance [--user admin_user] [--password admin_password] [--host
admin_host] [--port admin_port] [--local=true/false] [--domain domain_name]
[--debug=true/false] [--passwordfile file_name] [--secure | -s] instance_name
```

```
stop-instance [--user admin_user] [--password admin_password] [--host
admin_host] [--port admin_port] [--local=true/false] [--domain domain_name]
[--passwordfile file_name] [--secure | -s] instance_name
```

These commands have a `local` option which you can use to start or stop the server without going through the Administration Server. If you use the local option, you do not need to specify the `host`, `port`, `user`, and `password` (or `passwordfile`) options.

For information on the syntax of these commands, use the `asadmin` help. For information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Using the Windows Services (Windows)

You can start the server by using the Services Control Panel in Windows.

Follow these steps:

1. In the Control Panel click Administrative Tools.
2. Click Services.
3. Scroll through the list of services and double-click the service for your server.

   It is called Sun Application Server (*domain_name:instance_name*). For example, Sun Application Server (domain1:server1).
4. Click Start or Stop.
5. Click OK.

Using the startserv and stopserv Scripts

To use the `startserv` and `stopserv` scripts, at the command-line prompt go to the directory:

instance_dir / bin

where *install_dir* is the directory where you installed the server, *domain_dir* is the domain directory, and *instance_dir* is the name of the instance you want to start.

For UNIX, type:

```
./startserv
```

Log in as root if the server runs on ports with numbers lower than 1024; otherwise, log in as root or with the server's user account.

You can use the optional parameter `-i` at the end of the line. `The -i` option runs the server in `inittab` mode, so that if the server process is ever killed or crashed, `inittab` will restart the server for you. This option also prevents the server from putting itself in a background process.

NOTE	If the server is already running, the `startserv` command will fail. You must stop the server first, then use the `startserv` command. Also, if the server startup fails, you should kill the process before trying to restart it.

For Windows, type:

```
startserv
```

To stop the server manually, at the command-line prompt go to the directory:

instance_dir/bin

where *install_dir* is the directory where you installed the server and *instance_dir* is the name of the instance you want to start.

For UNIX, type:

```
./stopserv
```

If you used the `/etc/inittab` file to restart the server you must remove the line starting the server from `/etc/inittab` and type `kill -1 1` before you try to stop the server. Otherwise, the server restarts automatically after it is stopped.

For Windows, type:

```
stopserv
```

Starting the Application Server Instance in Debug Mode

You can run the application server instance in debug mode if developers want to debug their J2EE applications.

To start the server in debug mode:

1. Access the Administration interface and click the name of the application server instance you want to start in debug mode.
2. Click the General tab.
3. Click the checkbox next to Run in Debug Mode.
4. Restart the application server instance.

Debug mode changes the JVM settings. Debug Enabled is set to true, and the Debug Options change. For more information on the JVM Debug options, see the Java Platform Debugger Options documentation at http://java.sun.com/products/jpda/doc/conninv.html.

To start the application server instance in debug mode from the command-line interface, use the `asadmin` utility's `start-instance` command with the `debug` option set to true. For more information on the command syntax, see the online help for the command-line interface.

Setting the Termination Timeout

When you stop an application server instance, it stops accepting new connections. Then it waits for all outstanding connections to complete. The time the server waits before timing out is configurable in the `init.conf` file, which can be found in *instance_dir*`/config/`. By default it is set to 30 seconds. To change the value, add the following line to `init.conf`:

```
TerminateTimeout seconds
```

where *seconds* represents the number of seconds the server will wait before timing out.

The advantages to configuring this value is that the server will wait longer for connections to complete. However, because servers often have connections open from nonresponsive clients, increasing the termination timeout may increase the time it takes for the server to shut down.

Restarting an Application Server Instance Automatically (UNIX)

You can restart an application server instance using one of the following methods:

- Automatically restart it from the `/etc/inittab` file.

  Note that if you are using a version of UNIX not derived from System V, you will not be able to use the `/etc/inittab` file.

- Automatically restart it with daemons in `/etc/rc2.d` when the machine reboots.
- Restart it manually. See "Starting and Stopping an Application Server Instance" on page 69 and "Deleting an Application Server Instance" on page 78.

This section contains the following topics:

- About Restarting Automatically
- Restarting Automatically with /etc/inittab (UNIX)
- Restarting Automatically with the System RC Scripts (UNIX)

About Restarting Automatically

Because the installation scripts cannot edit the `/etc/rc.local` or `/etc/inittab` files, you must edit those files with a text editor. If you do not know how to edit these files, consult your system administrator or system documentation.

Normally, you cannot start an SSL-enabled server with either of these files because the server requires a password before starting. Although you can start an SSL-enabled server automatically if you keep the password in plain text in a file, this is *not* recommended.

CAUTION Leaving the SSL-enabled server's password in plain text in the server's startserv script is a large security risk. Anyone who can access the file has access to the SSL-enabled server's password. Consider the security risks before keeping the SSL-enabled server's password in plain text.

The server's `startserv` script, key pair file, and the key password should be owned by root (or, if a non-root user installed the server, that user account), with only the owner having read and write access to them.

Restarting Automatically with /etc/inittab (UNIX)

To restart the server using `inittab`, put the following text on one line in the `/etc/inittab` file:

```
http:2:respawn:install_dir/path_to_domain_dir/instance_dir/bin/startserv
-start -i
```

where *install_dir* is the directory where you installed the server, *path_to_domain_dir* is the path to the domain and *instance_dir* is the server's directory.

The `-i` option prevents the server from putting itself in a background process.

You must remove this line before you stop the server, otherwise the server automatically restarts.

Restarting Automatically with the System RC Scripts (UNIX)

If you use `/etc/rc.local`, or your system's equivalent, place the following line in `/etc/rc.local`:

install_dir`/`*path_to_domain_dir*/*instance_dir*`/bin/startserv`

Replace *install_dir* with the directory where you installed the server, *path_to_domain_dir* with the path to the domain, and *instance_dir* with the name of the application server instance.

Restarting an Application Server Instance Manually (UNIX)

On UNIX, you have the option of restarting the server instance manually. Unlike stopping the server instance, then starting it, a restart does not stop the watchdog program. For information about the watchdog, see "About the Watchdog" on page 77.

NOTE If you have made manual changes to your configuration files by editing them, you must apply changes before your restart the server, either by using the Apply Changes button in the Administration interface, or by using the `asadmin reconfig` command with the `keepmanualchanges` option set to `true`. For more information on applying changes, see "Applying Changes to an Application Server Instance" on page 79.

There are three ways to restart the server instance, covered in the following topics:

- Restarting the Server Instance Using the Restart Button (UNIX)
- Restarting the Server Instance Using the restart-instance Command (UNIX)
- Restarting the Server Instance Using the restartserv Script (UNIX)

Restarting the Server Instance Using the Restart Button (UNIX)

To restart the server instance using the Administration interface:

1. In the left pane, under App Server Instances, click the name of the instance to restart.
2. In the right pane, click Restart.
3. You see a message when the application server instance is restarted successfully.

Restarting the Server Instance Using the restart-instance Command (UNIX)

Using the command-line interface utility `asadmin`, you can start and stop your application server instances either from the command-line or from a script. Use the commands `restart-instance`. This command has the following syntax:

`restart-instance [--user` *admin_user*`] [--password` *admin_password*`] [--host` *admin_host*`] [--port` *admin_port*`] [--local=`*true/false*`] [--domain` *domain_name*`] [--passwordfile` *file_name*`] [--secure | -s]` *instance_name*

This command has a `local` option which you can use to restart the server instance without going through the Administration Server.

For information on the syntax of these commands, use the `asadmin` help. For information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Restarting the Server Instance Using the restartserv Script (UNIX)

To use the `restartserv` script, at the command-line prompt go to the directory:

instance_dir/bin

where *install_dir* is the directory where you installed the server, *domain_dir* is the domain directory, and *instance_dir* is the name of the instance you want to start.

Type:

```
./restartserv
```

Log in as root if the server runs on ports with numbers lower than 1024; otherwise, log in as root or with the server's user account.

About the Watchdog

The watchdog (`appserv-wdog` on UNIX, `appservd-wdog.exe` on Windows) is a program that is shipped with your Sun ONE Application Server. It performs the following tasks:

- Starts the server
- Stops the server
- If SSL/TLS is enabled, prompts the administrator for the trust database password when the server starts
- Restarts the server if it goes down

The watchdog runs in the background, without user intervention. You should never have to configure or otherwise change it. One watchdog runs for each application server instance, including the Administration Server.

On UNIX, each watchdog spawns a process for the primordial application server (`appservd`) process, which in turn spawns the `appservd` process that accepts requests. Since it starts the server, the watchdog process ID is shown in the `pid` log file in *instance_dir*/logs.

NOTE `appservd` Processes on UNIX Platforms: Although you will notice that there is a single appservd process started for each application server instance on Windows, there are two appservd processes started per application server instance on UNIX systems.

On UNIX, one appservd process is referred to as the "primordial" process while the second appservd process is referred to as the "worker" process. The worker process is the process that carries out the actual processing of application requests while the primordial process acts as an overarching controller. In a future release of the application server, you will have the option to define the number of worker processes for each application server instance. In the initial release of the product, only one worker process is supported per application server instance.

Adding an Application Server Instance

To add an application server instance using the Administration interface:

1. Access the Administration interface and click App Server Instances in the left pane.
2. Click the General tab.
3. On the Application Server Instances page, click New.
4. On the Create New Instance page, provide an instance name and a port number.

   The instance name must be unique for this Administration Server and domain. The port number must not be used by any other process on the machine.

   If you are using UNIX, you can also specify a UNIX user for the instance to run as.
5. Click OK.

For more information, see the online help.

To add another application server instance using the command-line interface, use the `asadmin` utility's `create-instance` command, which has the following syntax:

```
asadmin create-instance [--user admin_user] [--password admin_password]
[--host host] [--port port] [--sysuser sys_user] [--domain domain_name]
[--local=true/false] [--passwordfile file_name] [--secure | -s]
--instanceport instance_port instance_name
```

This command has a `local` option which you can use to restart the server instance without going through the Administration Server. The `sysuser` option is only for UNIX.

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Deleting an Application Server Instance

You can delete an application server instance from your administrative domain. Be sure that you don't need the application server instance anymore before you delete it, since this process cannot be undone.

To delete an application server instance from your machine using the Administration interface:

1. Access the Administration interface and click the name of the application server instance you want to remove.
2. Click the General tab.
3. Click Delete.

For more information, see the online help.

To delete an application server instance from your machine using the command-line interface, use the `asadmin` utility's `delete-instance` command, which has the following syntax:

```
asadmin delete-instance [--user admin_user] [--password admin_password]
[--host admin_host] [--port admin_port] [--domain domain_name]
[--local=true/false] [--passwordfile file_name] [--secure | -s] instance_name
```

This command has a `local` option which you can use to delete the server instance without going through the Administration Server.

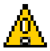
For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Applying Changes to an Application Server Instance

When you change configuration information using the Administration interface or the command-line interface, the changes are not applied immediately, but are saved into special files, located in *server_instance* / config / backup. The Administration interface and command-line interface display configuration values stored in files in the above directory. Until the changes you make are applied they do not take effect. Applying changes is also called reconfiguring the server. When you apply your changes, all changes made to the configuration since the last time you applied changes take effect. Note that restarting the instance does not apply the changes automatically.

If you've made changes to the server instance configuration that require you to apply changes, a yellow icon appears next to the application server instance in the left pane's tree view, in the banner when you access the server instance, and on the server instance's main page.

Figure 4-1 Warning Icon



To apply changes to an application server instance using the Administration interface:

1. Access the Administration interface and click the name of the application server instance you want to reconfigure.
2. Click the General tab.
3. Click Apply Changes.

   When the changes are applied, the screen displays a message.

To reconfigure an application server instance using the command-line interface, use the `asadmin` utility's `reconfig` command, which has the following syntax:

```
asadmin reconfig --user admin_user [--password admin_password] [--host
admin_host] [--port admin_port] [--passwordfile file_name] [--secure | -s]
[--discardmanualchanges=true/false | --keepmanualchanges=true/false]
instance_name
```

If you have made manual changes to the configuration files by editing them by hand, you must use `keepmanualchanges=true` to keep those edits during the reconfiguration (the option defaults to false). If you set `discardmanualchanges=true`, you discard any changes made manually. Setting `discardmanulchanges=false` (the default) does not mean the same thing as `keepmanualchanges=true`. Instead, setting it to false it is the equivalent of not specifying the `discardmanualchanges` option.

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

For some properties, you need to restart the server, as well as apply changes, in order for your changes to take effect. These properties include all properties set in the configuration files `init.conf` and `obj.conf`, and some properties in `server.xml`. For information about these files, see the *Sun ONE Application Server Administrator's Configuration File Reference.*

Your server will warn you if the changes require restart by putting a yellow warning icon next to the server instance left pane's tree view, in the banner when you access the server instance, and on the server instance's main page. The messages in the banner and on the page indicate that a restart is required. Once you restart the server instance, the yellow warning icons disappear.

The `server.xml` settings that do *not* require a restart include the following:

- Deploying, undeploying, and redeploying J2EE applications (EAR files), EJB modules (JAR files), web modules (WAR files), connectors (RAR files). Note that these settings also don't require Apply Changes.
- Enabling and disabling J2EE applications (EAR files), EJB modules (JAR files), web modules (WAR files), and connectors (RAR files).
- Creating, updating, and deleting resources.
- Setting monitoring enabled to true/false for the EJB container or MDB container.
- Changes to HTTP and web-container features (that is, in `server.xml`, changes to `http-service` and `web-container` and their sub-elements).

Viewing Application Server Instance Status

You can view whether a server is started or stopped, as well as basic application server instance settings, using the Administration interface.

To view application server instance status:

1. In the left pane, click the application server instance name.
2. In the right pane, click the General tab.

   You see whether the server is running or not running, as well as the hostname, port number, installation directory, and the version of the Sun ONE Application Server software.

To view the application server instance's status using the command-line interface, use the `asadmin` utility's `show-instance-status` command. The status is starting, started, stopping, or stopped. The command has the following syntax:

```
asadmin show-instance-status --user admin_user [--password
admin_password] [--host admin_host] [--port admin_port] [--passwordfile
file_name] [--secure | -s] instance_name
```

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, “Using the Command Line Interface.”

Configuring JVM Settings

You can configure Java Virtual Machine (JVM) settings for you application server instance. These settings include the location of your Java home, compiler options, debugging options, and profiler information. One reason to configure these settings is to improve performance. For more information on performance see the *Sun ONE Application Server Performance and Tuning Guide*.

This section describes the following topics:

- Configuring General Settings
- Configuring Path Settings
- Configuring JVM Options
- Configuring the JVM Profiler
- Configuring JVM Settings Using the Command-Line Interface

Configuring General Settings

To configure the JVM’s general options in the Administration interface:

1. In the left pane, click the application server instance name.
2. In the right pane, click the JVM tab.
3. Click General.
4. Set the Java Home.

   The Java Home is the path to the directory where the Java Developer’s Kit (JDK) is installed. Sun ONE Application Server supports the Sun JDK 1.4.0_02 or higher.

5. Choose whether to enable debugging and set debug options.

   A list of debug options is available at http://java.sun.com/products/jpda/doc/conninv.html#Invocation.

6. Choose rmic options.

   The rmic options field shows the rmic options passed to the RMI compiler at application deployment time. The `-keepgenerated` option saves generated source for stubs and ties. For more information about the rmic command, see the *Sun ONE Application Server Developer's Guide to Enterprise Java Beans*.

7. Click Save.

Configuring Path Settings

To configure the JVM's path settings in the Administration interface:

1. In the left pane, click the application server instance name.
2. In the right pane, click the JVM tab.
3. Click Path Settings.
4. Choose a suffix for the system's classpath.
5. Choose whether to ignore the environment classpath.

   If you do not ignore the classpath, the CLASSPATH environment variable is read and appended to the Sun ONE Application Server classpath. The CLASSPATH environment variable is added after the classpath-suffix, at the very end.

   For a development environment, the classpath should be used. For a production environment, this classpath should be ignored to prevent environment variable side effects.

6. Set a native library path prefix and suffix.

   The native library path is the automatically constructed concatenation of the Application Server installation relative path for its native shared libraries, the standard JRE native library path, the shell environment setting (LD_LIBRARY_PATH on UNIX), and any path specified in the profiler element. Since this is synthesized, it does not appear explicitly in the server configuration.

7. Click Save.

Configuring JVM Options

To set JVM command-line options in the Administration interface:

1. In the left pane, click the application server instance name.
2. In the right pane, click the JVM tab.
3. Click JVM Options.
4. To add a JVM option, type it in the text field at the top of the screen and click Add.
5. To delete a JVM option, click the checkbox next to it and click Delete.
6. To edit a JVM option, edit the text in the JVM Option field and click Save.

For information about specific JVM options, see http://java.sun.com/docs/hotspot/VMOptions.html

Configuring the JVM Profiler

To configure the JVM Profiler in the Administration interface:

1. In the left pane, click the application server instance name.
2. In the right pane, click the JVM tab.
3. Click Profiler.
4. Specify the name of the profiler, its classpath and native library path, and whether it is enabled.
5. To add a JVM option for the profiler, type it in the text field at the top of the screen and click Add.
6. To delete a JVM option for the profiler, click the checkbox next to it and click Delete.
7. To edit a JVM option for the profiler, edit the text in the JVM Option field and click Save.

For more information about profilers, see the *Sun ONE Application Server Developer's Guide*.

Configuring JVM Settings Using the Command-Line Interface

To configure JVM Settings using the command-line interface's `asadmin` utility, use the following commands:

To get all the attributes from an instance:

```
asadmin> get server_instance.java-config.*
```

To get an attribute called `classpathprefix` in server1:

```
asadmin> get server1.java-config.classpathprefix
```

To set an attribute called `classpathprefix` in server1:

```
asadmin> set server1.java-config.classpathprefix=com.sun
```

The above examples all assume you have already set the user, password, host, and port in your environment variables. For a full list of attributes, see Appendix A, "Using the Command Line Interface."

To set JVM options using the command-line interface's `asadmin` utility, use the following commands:

```
asadmin> create-jvm-options --user admin_user [--password admin_password]
[--host host] [--port port] [--secure | -s] [--instance instance_name]
[--profiler=true/false]
(jvm_option_name=jvm_option_value)[:jvm_option_name=jvm_option_name]*
```

```
asadmin> delete-jvm-options --user admin_user [--password admin_password]
[--host host] [--port port] [--secure | -s] [--instance instance_name]
[--profiler=true/false]
(jvm_option_name=jvm_option_value)[:jvm_option_name=jvm_option_name]*
```

Note: you can enter more than one JVM option separated by colon. If the options are used by the profiler, set `--profiler` to `true`.

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Configuring Logging Setting and Monitoring Settings

The settings on the Logging and Monitoring tabs are Logging and Monitoring settings are covered in separate chapters. For information on logging, see Chapter 5, "Using Logging." For information on Monitoring and SNMP settings, see Chapter 6, "Monitoring the Sun ONE Application Server."

Changing Application Server Instance Advanced Settings

An application server instance has additional settings showing the instance's locale (which determines settings such as the character set and language), the path to the server's log files, the path to the directory for deployed applications, and the path to the session store directory where passivated beans and persistent HTTP sessions are stored.

In addition, you enable application reloading and poll interval for how often to reload. Dynamic application reloading automatically checks applications for changes, and serves the updated version automatically if they have been changed. In general, you should enable dynamic reloading in a development environment and not in a production environment. The poll interval specifies the interval at which the Application Server checks the applications for updates.

To change the application server instance's settings using the Administration interface:

1. In the left pane, click the application server instance name.
2. On the application server instance's page, click the Advanced tab.
3. Enter the desired value in the fields.
4. Click Save.

To change the server instance's advanced setting using the command-line interface's `asadmin` utility, you use the get and set commands. When you get all the attributes for a server instance

To get all the attributes from an instance:

`asadmin get` *instance_name*`.*`

For example:

```
asadmin get server1.*
```

To get an attribute called `logRoot` for `server1`:

```
asadmin get server1.logRoot
```

To set an attribute called `logRoot` for `server1`:

```
asadmin set server1.logRoot=/space/log
```

The above examples all assume you have already set the user, password, host, and port in your environment variables. For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Chapter 5

Using Logging

This chapter describes the logging features and functions of the Sun ONE Application Server. In addition, components for which logging may be used are discussed.

This chapter includes the following topics:

- About Logging
- Logging on the UNIX and Windows Platform
- Using Log Levels
- About Virtual Servers and Logging
- About Loggers
- About Client Side Logging
- Redirecting Application and Server Log Output
- Log File Management
- Configuring Logging Through the Command-line Interface
- Configuring Logging Through the Administration Interface
- Configuring the Directives for Error Logging
- Viewing the Access Log File
- Viewing the Event Log Filew
- Setting Log Preferences
- Running the Log Analyzer
- Viewing Events (Windows 2000 Pro)

About Logging

Logging is a useful debugging and diagnostic tool when used in applications. It also increases the developer's productivity. The application server's own log output can help identify and diagnose server configuration and deployment problems.

Logging within the Sun ONE Application Server uses the Java logging API. Sun ONE Application Server collects and stores logging information in two log files, namely, `access.log` and `server.log` located in the `logs` directory. You can also direct logs to your own log file.

A logged message provides more information than simply the message itself. The additional information provided includes:

- Date and time of the event.
- Log level for the event. Appserver specified Loglevel ID or name.
- Process ID (PID). PID of the appserv process.
- (Optional) Virtual server ID (vsid). The vsid that generated the message.
- Message ID. Subsystem and a four digit integer.
- Message data.

The type and order of the additional message information varies depending upon the platform used for logging and the logging service enabled for that platform. To enable the virtual server ID for logged messages, see "Configuring the Log Service" on page 113.

Logging on the UNIX and Windows Platform

This section discusses how log files are created. In addition, this section includes the following topics:

- Default Logging in server.log
- Logging Using syslog
- Logging Using the Windows eventlog

Default Logging in server.log

On both the UNIX and Windows platforms, the log files are created in `server.log` located in the `log` sub-directory. Logs coming from all the server components and virtual servers of an instance are collected in this single file.

The default log level for the entire server can be set. However, you can also override the default log level for a particular subsystem at the subsystem level. You can also redirect `stdout` and `stderr` to the server's event log and direct the log output to the operating system's system log. Additionally, you can direct `stdout` and `stderr` content to the server's event log. Log messages by default are sent to `stderr` in addition to the specified server log file.

Another feature available is to log the virtual server ID with the log message. This is a useful feature when multiple virtual servers are used to log messages to the same log file. You can choose to write the log messages to system log. When you do so, logging is not performed on the `server.log` file. Instead the `syslog` logging service on UNIX, or the system logging service on Windows platform is used to produce and manage logs.

You can also use the `server.xml` attributes to control the contents of this file. For details about the `server.xml` file, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Example of server.log

The following is an example of `server.log`.

Timestamp, Log Level, (PID vsid(optional)): messageID: message

```
[01/Aug/2002:11:39:31] INFO ( 1224): CORE1116: Sun ONE Application
Server 7.0

[01/Aug/2002:11:39:36] INFO ( 1224): CORE5076: Using [Java
HotSpot(TM) Server VM, Version 1.4.0_02-20020712] from [Sun
Microsystems Inc.]

[01/Aug/2002:11:39:50] INFO ( 1224): JMS5023: JMS service
successfully started. Instance Name = domain1_server1, Home =
[D:\install_7_29\imq\bin].

[01/Aug/2002:11:39:53] INFO ( 1224): CIS0056: Creating TCP
ServerConnection at [EndPoint
[IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]

[01/Aug/2002:11:39:53] INFO ( 1224): CIS0057: Created TCP
ServerConnection at [EndPoint
[IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]
```

```
[01/Aug/2002:11:39:54] INFO ( 1224): CIS0054: Creating TCP
Connection from [-] to [EndPoint
[IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]
```

NOTE Redirecting log messages pertaining to pre-compiled JSPs:

The log messages pertaining to the JSPs that are pre-compiled are stored by default in the administration server's log file which is located at `{`*domain_root*`}/{`*domain_name*`}/admin-server/logs/server.log`.

Since all the messages are logged to the same file, exceptions or errors thrown while deploying applications using pre-compiled JSPs could get lost in the volume of messages in the common log file. When multiple applications are being deployed to multiple instances under a given domain, the log messages in the admin-server will need to be carefully scrutinized for any exceptions relevant to a particular application's JSPs. This causes redundancy.

Therefore, it is better to log messages pertaining to applications that are deployed using pre-compiled JSPs, in the server instance’s server.log file, instead of the administrative server’s server.log file.

To redirect the log messages to your Sun ONE Application Server instance’s server.log file, change the path to the log file in the Administrator Interface. See Configuring the Log Service for more information.

Logging Using syslog

For stable operational environments where centralized logging is required `syslog` is appropriate. For environments where log output is frequently required for diagnostics and debugging, individual server instance or virtual server logs may be more manageable.

NOTE

- All logged data for the server instance and administration server in one file may prove difficult to read and debug. It is recommended that you use the syslog master log file only for deployed applications that are running smoothly.
- Logged message are intermixed with all other logs from the Solaris daemon applications.

By using the `syslog` log file, in conjunction with `syslogd`, and the system log daemon, you can configure the `syslog.conf` file to:

- Log messages to the appropriate system log

- Write messages to the system console
- Forward logged messages to a list of users, or forward logged messages to another syslogd on another host over the network

---

NOTE Following installation of Sun ONE Application Server, the log service element attribute for the server, use-system-logging, is not enabled. This means that logs by default are not directed to syslog on UNIX or the Windows Event Log on Windows platforms.You can direct logging to syslog or the Windows Event Log by enabling this attribute in the Server element of server.xml described in the Sun ONE Application Server Configuration File Reference. Before setting use-system-logging, see "Log File Management," on page 107.

Configuring syslog

To allow better manageability and readability, you can direct messages with less severity to be stored in a separate file by configuring syslog.conf located in the /etc directory.

To configure syslog:

1. To direct messages with less severity to be stored in a separate file add the following command to the syslog.conf file in Solaris:

```
daemon.debug  /var/adm/iasdebug
```

NOTE When log messages are directed to the Windows eventlog, only messages with level INFO, WARNING, SEVERE, ALERT or FATAL are logged.

---

2. Give a hang-up signal to syslogd. This can be done by using the following command:

```
kill -HUP <PID syslogd>
```

3. Go to the Admin Server in the Administration interface and select the Write to system log option. Save, and apply changes. Restart the Admin Server for the changes to take effect.

   An example of a configured Solaris `syslog.conf` file follows:

```
#ident"@(#)syslog.conf1.598/12/14 SMI"/* SunOS 5.0 */

#

# Copyright (c) 1991-1998 by Sun Microsystems, Inc.

# All rights reserved.

#

# syslog configuration file.

#

# This file is processed by m4 so be careful to quote (`') names

# that match m4 reserved words.  Also, within ifdef's, arguments

# containing commas must be quoted.

#

*.err;kern.notice;auth.notice/dev/sysmsg

*.err;kern.debug;mail.crit/var/adm/messages

daemon.info;daemon.err;daemon.debug;daemon.alert;daemon.crit;dae
mon.warning/var/adm/iaslog

daemon.debug/var/adm/iasdebug

#daemon.notice;/var/adm/iaslognotice

#daemon.warning;/var/adm/iaslogwarning

#daemon.alert;/var/adm/iaslogalert

#daemon.err;/var/adm/iaslogerr


#*.alert;kern.err;daemon.erroperator

#*.alertroot

*.emerg*
```

```
# if a non-loghost machine chooses to have authentication
messages
# sent to the loghost machine, un-comment out the following line:
#auth.noticeifdef(`LOGHOST', /var/log/authlog, @loghost)

mail.debugifdef(`LOGHOST', /var/log/syslog, @loghost)
#
# non-loghost machines will use the following lines to cause
"user"
# log messages to be logged locally.
#
ifdef(`LOGHOST', ,
user.err/dev/sysmsg
user.err/var/adm/messages
user.alert`root, operator'
user.emerg*
)
```

For more information, see the `syslog.conf` man page.

Any change made to `syslog.conf` will require a restart of the Sun ONE Application Server for the change to take effect.

Since logging to `syslog` means, logs from all Sun ONE Application Servers, and other daemon applications are collected in the same file, logged messages are enhanced with the following information to identify Sun ONE Application Server-specific messages from the particular server or virtual server instance:

- Unique message ID
- Timestamp
- Instancename
- Program name (`appservd` or `appserv-wdog`)
- Process ID (PID of the appserv process)
- Thread ID (optional)

- Server ID

The log service can be configured for both the server instance and the virtual server instance in the `server.xml` file. The log service configuration for the virtual server instance is described in the section "About Virtual Servers and Logging" on page 100. The log service configuration for the server instance is described in "Configuring Logging Through the Administration Interface" on page 112.

Log levels are configurable through the Administration interface for applicable subsystems and components.

For more information on the `syslog` logging mechanism used in the UNIX operating environment, use the following man commands at a terminal prompt:

```
man syslog
```

```
man syslogd
```

```
man syslog.conf
```

Example of syslog messages

The following is an example of `syslog` messages.

Timestamp, hostname [instance_name], [subsystem], [vsid], Message ID, Loglevel, Message data

```
Jul 19 14:33:18 strange /usr/lib/nfs/lockd[164]: [ID 599441
daemon.info] Number of servers not specified. Using default of 20.
```

```
Jul 19 14:33:20 strange ntpdate[181]: [ID 558275 daemon.notice]
adjust time server 192.18.56.149 offset 0.06702 6 sec
```

```
Jul 19 14:38:13 strange xntpd[248]: [ID 204180 daemon.info]
synchronisation lost
```

```
Jul 19 14:38:47 strange server1 appservd[374]: [ID 702911
daemon.info] INFO (  374): CORE1116: Sun ONE Application Server 7.0
```

```
Jul 19 14:38:48 strange server1 appservd[374]: [ID 702911
daemon.info] FINE (  374): Collecting statistics for up to 1
processes with 128 threads, 200 listen sockets, and 1000 virtual
servers
```

Logging Using the Windows eventlog

For more information on the event log mechanism used in the Windows operating environment, refer to the Windows help system index for the keywords *Event Logging*.

Using Log Levels

This section discusses log levels and how you can assign a log level for each Sun ONE Application Server subsystem.

The following topics are described:

- About Log Levels
- Log Levels Used for syslog Configuration

About Log Levels

The Sun ONE Application Server uses the standard JDK 1.4 log levels for selective logging of information. In addition to the standard JDK log levels, Sun ONE Application Server has added log levels designed to map more intuitively to `server.log` and to tightly integrate with Solaris.

When logged messages are routed to `server.log`, they are also mapped to log levels as defined in the "Sun ONE Application Server Log Levels Mapped to server.log" on page 99.

NOTE The default log level for the `server.log` file (or for `syslog`) for the Admin Server and the default application server instance is INFO. When the default log level is used for the application server instance, error and information messages are logged. To avoid logging such messages, change the log level to WARNING or SEVERE in the `server.xml` file or in the Administration interface of the Admin Server and the server instance.

Default log level across the server can be set in the log-service element. This affects any element that has log level set to "default".

You can assign a log level for each Sun ONE Application Server subsystem for which logging is enabled. A log level is useful to streamline the amount of message information that is recorded during runtime. The level is specified in the `server.xml` file for the intended subsystem. You can specify the log level from the Administration interface for the selected subsystem, or, you can edit the `server.xml` file directly to set the desired log level for the selected subsystem.

CAUTION Manual edits to the `server.xml` file may introduce syntax errors that result in server startup failures. Guidelines for manually editing configuration files are discussed in the *Sun ONE Application Server Administrator's Configuration File Reference*, section "Manually Editing Configuration Files".

An example of setting the log level through the Administration interface is shown in the figure "Log Level for JMS Service" on page 102. To set the log level for each subsystem or component directly in the `server.xml` file, see the *Sun ONE Application Server Administrator's Configuration File Reference.*

The Log levels described in the table "Log Levels" on page 98 match the requirements of the JDK1.4 logging API specification. However, the log levels ALERT and FATAL are specific to Sun ONE Application Server and are not implemented in the JDK1.4 logging API.

The following table defines the log levels and messages in Sun ONE Application Server, in increasing order of severity. The left column lists the log level designation in the Sun ONE Application Server and the right column provides a brief description of each log level.

Table 5-1 Log Levels

Log level	Description
FINEST FINER FINE	Messages indicate extent of verbosity of debug messages. FINEST gives the maximum verbosity.
CONFIG	Messages relate to a variety of static configuration information, to assist in debugging problems that may be associated with particular configurations.
INFO	Messages are informative in nature, usually related to server configuration or server status. These messages do not indicate errors that need immediate action. For example, a message could be logged that configuration change notification has been received; creating a new topic on MessageBroker
WARNING	Messages indicate a warning. The message would probably be accompanied by an exception.
SEVERE	Messages indicate an event of considerable importance that is likely to prevent normal application execution
ALERT*	Messages alert the user to take specific action.

Table 5-1 Log Levels

Log level	Description
FATAL*	Messages indicate a fatal error, after which server execution is not recommended. Ideally, this would be the last message before a server crash.

* Log levels specific to Sun ONE Application Server.

NOTE	All messages with a log level less than INFO (FINEST, FINER, FINE, and CONFIG) provide information that help with issues relating to debugging and must be enabled as advised by technical support. Messages with a log level less than INFO are generally not localized.

Log Levels Used for syslog Configuration

The following table contains a list of log levels that can be configured within the Sun ONE Application Server when using `syslog`. The left column lists the log level designation in the Sun ONE Application Server, and right column provides the corresponding log level in the `syslog` facility.

Table 5-2 Sun ONE Application Server Log Levels Mapped to server.log

Sun ONE Application Server	syslog level
FINEST	LOG_DEBUG
FINER	LOG_DEBUG
FINE	LOG_DEBUG
CONFIG	LOG_INFO
INFO (default)	LOG_INFO
WARNING	LOG_WARNING
SEVERE	LOG_ERR
ALERT	LOG_ALERT
FATAL	LOG_CRIT

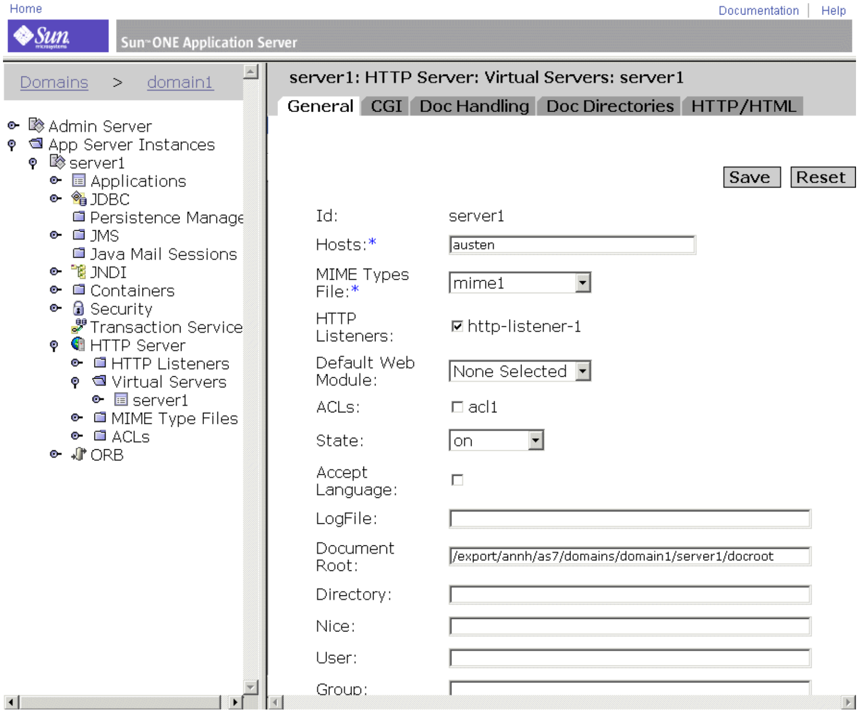
About Virtual Servers and Logging

The Sun ONE Application Server can have virtual server instances. Each virtual server within a Sun ONE Application Server instance has its own identity and may have its own log file. The use of separate log files for each virtual server can help track server activity for particular transactions and resources.

To specify the log file name for a virtual server from the Administration interface, go to the HTTP Server link from the directory tree and open the server instance element under the virtual server folder to display the General tab on the right frame. You can enter the path and name of the log file for this virtual server in the Log File field. The figure "Setting the Virtual Server Log File Name" on page 100 shows where this setting is located.

NOTE When logging is enabled and an application is running, logged messages from the application are logged without the virtual server ID.

Figure 5-1 Setting the Virtual Server Log File Name



server1: HTTP Server: Virtual Servers: server1

Id:	server1
Hosts:*	austen
MIME Types File:*	mime1
HTTP Listeners:	☑ http-listener-1
Default Web Module:	None Selected
ACLs:	□ acl1
State:	on
Accept Language:	□
LogFile:	
Document Root:	/export/annh/as7/domains/domain1/server1/docroot
Directory:	
Nice:	
User:	
Group:	

You can also direct logged messages from multiple virtual servers to one server log file. When so doing, you may wish to enable the `log-virtual-server-id` in the `log service` element of the `server.xml` file. This helps users to distinguish log messages originating from different virtual servers.

```
<log-service level="FINEST" log-stdout="false" log-stderr="false"
echo-log-messages-to-stderr="false" create-console="false"
log-virtual-server-id="true" use-system-logging="false">

</log-service>

   <http-listener>

      <virtual-server-class>

         <virtual-server id="server1"
http-listeners="http-listener-1" hosts="strange" mime="mime1"
state="on" accept-language="false"/>

         <virtual-server id="server2" hosts="strange"
mime="mime1"/>

      </virtual-server-class>

   </http-listener>
```

In this example, `<log-service log-virtual-server-id="true">` is responsible for including `virtual_server_id` in every log message. This allows you to differentiate messages coming from different virtual servers. The absence of attribute “log-file” in the virtual-server element, causes all the virtual servers to log messages to a single file.

About Loggers

Logging can be enabled or disabled selectively at the subsystem level. Logging control for each subsystem is specified in the `server.xml` file, as described in the *Sun ONE Application Server Configuration File Reference*. Each subsystem has its own logger in accordance with the requirements of the JDK1.4 logging API.

In the following table, the left column defines the subsystem and the right column the element in the `server.xml` file for each subsystem.

Table 5-3 Subsystems and Locations in Sun ONE Application Server

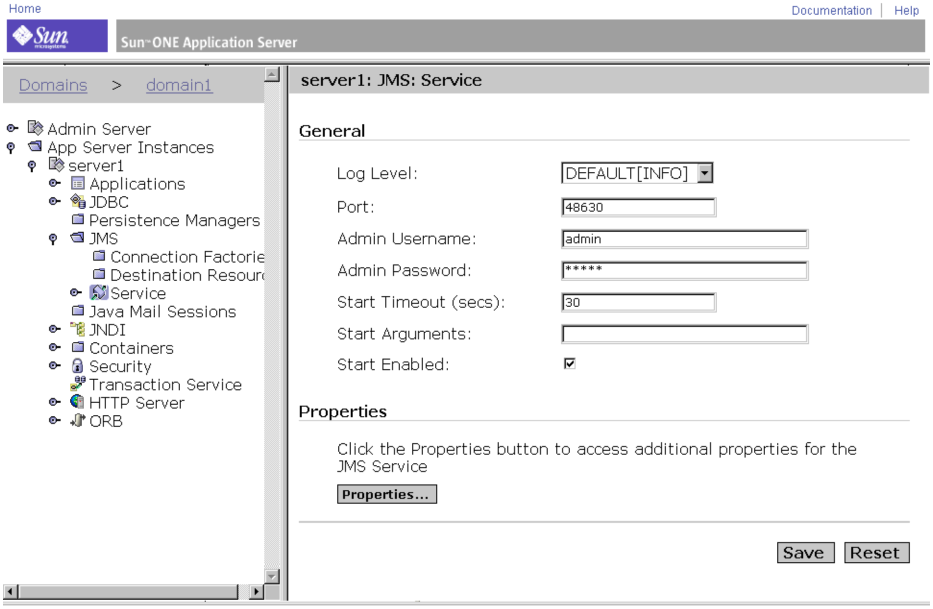
Subsystem	Element
Administration Server	<admin-service>
EJB Container	<ejb-container>

Table 5-3 Subsystems and Locations in Sun ONE Application Server

Subsystem	Element
Web Container	<web-container>
MDB Container	<mdb-container>
Sun ONE Message Queue (JMS service)	<jms-service>
Security Service	<security-service>
Java Transaction Service (JTS)	<transaction-service>
Object Request Broker (ORB)	<iiop-service>
Default handlers[1]	<log-service>

[1] Default handlers refers to the default logger associated with all `server.xml` entries that are not associated with a particular subsystem, such as the utility classes.

Figure 5-2 Log Level for JMS Service



server1: JMS: Service

General

Log Level:	DEFAULT[INFO]
Port:	48630
Admin Username:	admin
Admin Password:	*****
Start Timeout (secs):	30
Start Arguments:	
Start Enabled:	☑

Properties

Click the Properties button to access additional properties for the JMS Service

NOTE On the Windows platform, if you choose to send logs to the Windows `server.log`, only messages with level `INFO`, `WARNING`, `SEVERE`, `ALERT` or `FATAL` are logged to Windows Event Log.

The table "Log Levels" on page 98 defines the log levels, in increasing order of severity, provided for messages in Sun ONE Application Server. These log levels match the requirements of the JDK1.4 logging API specification. In addition, the log levels ALERT and FATAL are specific to Sun ONE Application Server and are not supported in the JDK1.4 logging API.

About Client Side Logging

The Application Client Container (ACC) has its own log service and can only log to a local file.

The ACC typically runs in its own process, on a different host from the application server. As such it has its own logging infrastructure and its own log file. The ACC configuration is held in the file `sun-acc.xml`.

The client subsystem logging element for the ACC is `log-service`. The following table lists the element, and the attributes, each with the noted default and range of values.

Table 5-4 ACC Logging Element

Element	Attribute	Description
`log-service`	file	ACC log file; when empty or missing, log to `stdout`.
`log-service`	level	ACC log level.

An example of the `sun-acc.xml` file is provided in the *Sun ONE Configuration File Reference*.

Redirecting Application and Server Log Output

For developers, it is important that the application logs and server logs be made readily available during unit testing for application components and J2EE applications. On the Windows platform, developers prefer to see server log messages displayed in a command window on the desktop. On the UNIX platform, many developers are comfortable with simply having the log messages stream to `stderr` in the terminal window in which the server instance is started, or, use the command tail `-f` to see the log messages written in log files.

The `server.xml` file contains attributes that can be set for `stdout` and `stderr` to direct logged messages to a log file or to the terminal window, and so forth. See the *Sun ONE Application Server Configuration File Reference* for more information on the use of `stdout` and `stderr`.

See "Configuring the Log Service" on page 113 for information on the log service.

Log File Management

You can set up your access and event log (`server.log`) files to be automatically archived. At a certain time, or after a specified interval, your logs are rotated. Sun ONE Application Server saves the old log files and stamps the saved file with a name that includes the date and time they were saved.

NOTE	Although you can create multiple virtual servers and associate a log file for each virtual server, log rotation settings for individual virtual servers is not supported.

For example, you can set up your access log files to rotate every hour, and Sun ONE Application Server saves and names the file "`access.199907152400`," where name of the log file, year, month, day, and 24-hour time is concatenated together into a single character string. The exact format of the log archive file varies depending upon which type of log rotation you set up.

NOTE These facilities are primarily provided for non-Solaris platforms.

For Solaris, these facilities are not enabled by default, and you must use the native Solaris operating system log management facilities such as `logadm` on Solaris 9. On Solaris 8, the preferred utility for log management would be the `cron` facility described in "Using the Solaris cron Utility to Schedule Execution of logadm" on page 111.

Depending on the operating system, there are four distinct ways you can perform log rotations. These are discussed in the section that follows. Topics include:

For UNIX and Windows:

- Internal-daemon Log Rotation
- Scheduler Based Log Rotation

For Solaris 9

- Using Solaris `logadm` utility. For more information, see "Rotation Using Solaris logadm Utility" on page 107.

For Solaris (any version)

- Using Solaris `cron` utility. For more information, see "Rotation Using Solaris "cron" Utility" on page 109.

Internal-daemon Log Rotation

Internal-daemon log rotation is available for both UNIX and Windows operating systems. Internal-daemon log rotation occurs within the HTTP daemon and can only be configured at the server instance startup time. Logs rotated using this method are saved in the following format:

```
access.<YYYY><MM><DD><HHMM>
```

```
error. <YYYY><MM><DD><HHMM>
```

You can specify the time used as a basis to rotate log files and start a new log file. For example, if the rotation start time is 12:00 a.m., and the rotation interval is 1440 minutes (one day), a new log file is created immediately upon save regardless of the present time and collects information until the rotation start time. The log file rotates every day at 12:00 a.m., and the access log is stamped at 12:00 a.m. and saved as access.199907152400. Likewise, if you set the interval at 240 minutes (four hours), the four-hour intervals begin at 12:00 a.m. such that the access log files will contain information gathered from 12:00 a.m. to 4:00 a.m., from 4:00 a.m. to 8:00 a.m., and so forth.

If log rotation is enabled, log file rotation starts at server startup. The first log file to be rotated gathers information from the current time until the next rotation time. Using the previous example, if you set your start time at 12:00 a.m. and your rotation interval at 240 minutes, and the current time is 6:00 a.m., the first log file to be rotated will contain the information gathered from 6:00 a.m. to 8:00 a.m, and the next log file will contain information from 8:00 a.m. to 12:00 p.m. (noon), and so forth.

Scheduler Based Log Rotation

Scheduler log rotation, allows you to archive log files immediately or have the server archive log files at a specific time on specific days. To archive log files immediately select Admin Server from the left pane of the Administration interface. Then, click on the Logging link at the top of the right page. Next, click Log Rotation. Finally, click Archive.

Logs rotated using the scheduler method are saved as the original filename followed by the date and time the file was rotated. For example, access might become access.24Apr-0430PM when it is rotates at 4:30 p.m.

Log rotation is initialized at server startup. If rotation is turned on, Sun ONE Application Server creates a time-stamped access log file and rotation starts at server startup.

Once the rotation starts, Sun ONE Application Server creates a new time stamped log file when there is a request or error that needs to be logged to the access or error log file and it occurs after the prior-scheduled “next rotate time”.

NOTE For the Windows platform, and for server logging directed to a file other than syslog on Solaris, you must archive the server logs.

To archive log files and to specify use of the schedulerd control method, select Admin Server from the left pane of the Administration interface. Then, click on the Logging link at the top of the right page. Next, click the Scheduler based Log Rotation box. Finally, OK. The current state of thescheduler is indicated.

Rotation Using Solaris logadm Utility

The Solaris 9 operating system incorporates the utility `logadm` which can be used to perform an array of functions with logged messages.

Particular to the Sun ONE Application Server, this utility is useful for performing log rotation tasks when running it from the Solaris cron utility, described in "Using the Solaris cron Utility to Schedule Execution of logadm" on page 111.

You can specify the following log rotation details with respect to log files:

- All log file names on the system that must be rotated
- Rotation interval
- Condition that will trigger the rotation
- Number of backup log files to be saved
- Naming convention of backup log files to be saved

The above details are specified in the file `logadm.conf` located in:

n /etc/logadm.conf

A sample `logadm.conf` file follows:

```
# Copyright 2001-2002 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)logadm.conf 1.2 02/02/13 SMI"
#
# logadm.conf
#
# Default settings for system log file management.
# The -w option to logadm(1M) is the preferred way to write to this
# file,
# but if you do edit it by hand, use "logadm -V" to check it for
# errors.
#
# The format of lines in this file is:
# <logname> <;options>
# For each logname listed here, the default options to logadm

# are given.  Options given on the logadm command line override

# the defaults contained in this file.

# # logadm typically runs early every morning via an entry in

# root's crontab (see crontab(1)).

#

/var/log/syslog -C 8 -P 'Tue Jul 9 10:10:00 2002' -a 'kill -HUP `cat

/var/run/syslog.pid`' /var/adm/messages -C 4 -P 'Tue Jul 30 10:10:00
2002' -a

'kill -HUP `cat /var/run/syslog.pid`' /var/cron/log -c -s 512k -t
/var/cron/olog

/var/lp/logs/lpsched -C 2 -N -t '$file.$N'

#

# The entry below is used by turnacct(1M)

#

/var/adm/pacct -C 0 -N -a '/usr/lib/acct/accton pacct' -g adm -m 664
-o adm -p never

#

# The entry below will rotate SUN One application server's default
logfile

# every day provided the current logfile size is >= 512k.  It will
compress

# the old log file before archiving it and also delete the old files
after 30

# days. The compression is done with gzip(1) and the resulting log
file has

# the suffix of .gz.

/var/appserver/domains/domain1/server1/logs/server.log -A 30d -s
512k -p 1d -z
```

Alternatively, you can start log rotation on a specific file by invoking the `logadm` command interactively.

The following example rotates `syslog` and keeps eight log files. Old log files are placed in the directory `/var/oldlogs` instead of `/var/log`:

```
% logadm -C8 -t'/var/oldlogs/syslog.$n' /var/log/syslog
```

You can also use an interactive command-line option to invoke rotation on a file specified in /etc/logadm.conf, but with different or modified options.

If options are specified both in /etc/logadm.conf and on the command-line, those in the /etc/logadm.conf file are applied first. Therefore, the command-line options override those in /etc/logadm.conf. An example of this is as follows:

```
% logadm /var/appserver/domains/domain1/server1/logs/server.log -p
now
```

The above command rotates the given file using all the options provided for that file in /etc/logadm.conf.

NOTE When multiple options are specified at a time, there is an implied AND between them. This means that all conditions must be met before the log is rotated.

For detailed information on logadm utility and its options, please refer to its manpage as follows:

```
% man logadm
```

OR

```
% logadm -h
```

Rotation Using Solaris “cron” Utility

On Solaris 8, the cron utility can be used to perform application server log rotations. This can be done by using the following command:

```
% crontab -e
```

This starts your favorite editor (defined by env. variable $EDITOR) so that you can provide the list of cron entries.

NOTE This command also invokes /etc/cron.d/logchecker script as soon you exit from the editor. This script feeds the changed/new crontab entries to cron daemon. Therefore, entries added this way are immediately picked up by cron daemon and log rotations starts immediately.

You need not restart the cron daemon to enable log rotations.

This section includes the following topics:

- About the crontab Entry Format
- Using the Solaris cron Utility to Schedule Execution of logadm

About the crontab Entry Format

A `crontab` file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five fields are integer patterns that specify the following:

- minute (0-59)
- hour (0-23),
- day of the month (1-31),
- month of the year (1-12),
- day of the week (0-6 with 0=Sunday).

Using this format, you can specify access and eventlog files to be rotated at specified time of day / week / month and schedule to repeat the rotations. For example,

0 0 * * 1-5

`/opt/SUNWappserver7/appserver/domains/domain1/server1/bin/rotatelogs`

0 12 * * 1-5

`/opt/SUNWappserver7/appserver/domains/domain1/server1/bin/rotatelogs`

0 * * * 1-5

`/opt/SUNWappserver7/appserver/domains/domain1/mainserver/bin/rotatelogs`

This rotates access and log files for server1 at midnight and noon every day between Monday and Friday; while access and log files for mainserver is rotated every hour of every day between Monday and Friday.

The crontab files are stored under `/var/spool/cron/crontabs/`. You can create the crontab file as an end user or root. Depending on your privileges, you can see the crontab entries by using the following command:

```
% crontab -l username
```

Using the Solaris cron Utility to Schedule Execution of logadm

The `cron` command starts a process that executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in `crontab` files in the directory `/var/spool/cron/crontabs`.

As an example of a regularly scheduled command used with `cron`, the following entry in `crontab` will start `logadm` everyday at midnight.

```
0 0 * * 0-6 logadm
```

Note that users can submit their own `crontab` file using the `crontab(1)` command.

To keep a log of all actions taken by `cron`, CRONLOG=YES (by default) must be specified in the `/etc/default/cron` file. `/etc/cron.d/logchecker` is a script that checks to see if the log file has exceeded the system `ulimit`. If so, the log file is moved to `/var/cron/olog`.

Configuring Logging Through the Command-line Interface

You can configure aspects of the logging service from the command-line for the server instance and for the virtual server instance.

NOTE All the command examples in this section assume that the environment variables have been set.

To get all of the log-service attributes for the server instance:

```
asadmin> get instance_name.log-service.*
```

The log-service attributes are also defined in the table “Log Service Attributes” on page 114.

An example of using this command with a designated server instance name follows:

```
asadmin> get server1.log-service.*
```

The list of attributes for the logging service of the server1 instance is returned. You can configure each listed attribute by using the `set` command.

To enable the logging of a virtual server ID for the virtual server instance, enter the following command at the terminal prompt:

```
asadmin> get instance_name.LogVirtualServerId
```

The current state of the LogVirtualServerId is returned. If the state is false, you can enable it with the `set` command as follows:

```
asadmin> set instance_name.LogVirtualServerId=true
```

To set the log file name for a virtual server instance, use the `set` command as follows:

```
asadmin> set instance_name.virtual-server.<virtual server
id>.logFile=<log file>
```

As an example, the following set log file command is issued:

```
asadmin> set
instance2.virtual-server.instance2.logFile=/space/IAs7se/appserver7/appse
rv/domains/domain1/instance2/logs/log
```

For more information on command syntax, see the command-line interface help.

For more information on using `asadmin`, see Appendix A, “Using the Command Line Interface.”

Configuring Logging Through the Administration Interface

This section describes the tasks you can perform through the Sun ONE Application Server Administration interface to configure the available logging service options for both server-wide (global) elements, directives, and application components.

This section includes the following topics:

- Configuring the Log Service
- Configuring Logging for Application Server Components and Subsystems
- Configuring the Directives for Error Logging

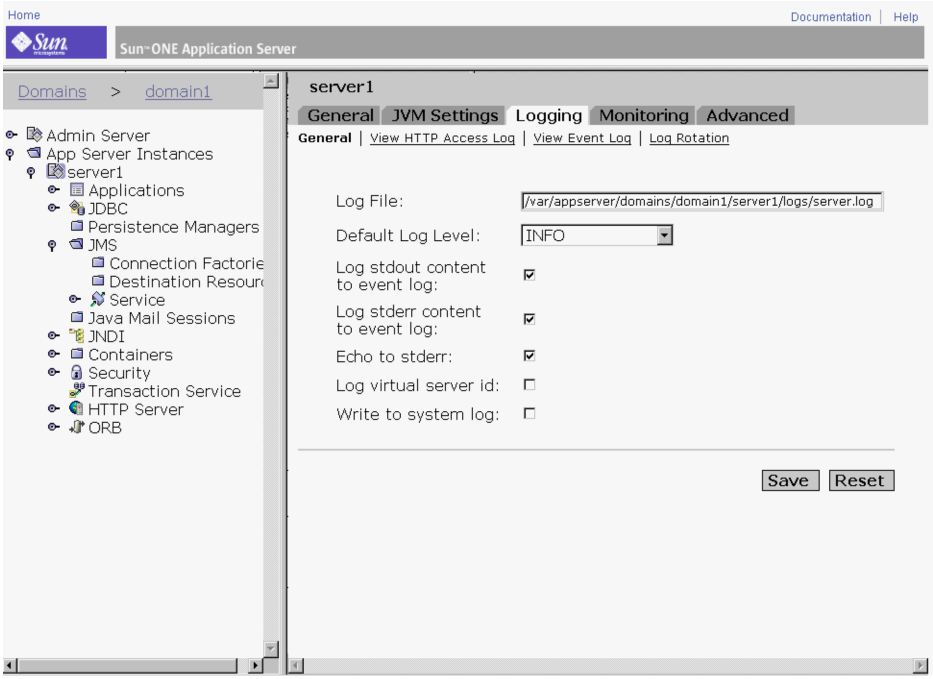
Configuring the Log Service

The log service is an element within the J2EE Service Element category in the `server.xml` file, as described in the *Sun ONE Application Server Configuration File Reference*. The log service is used to configure the system logging service, which includes the following log files:

- Server log
- Access log
- Transaction log
- Virtual server log

Configuration of the system logging service includes specifying values for attributes of the log service element.

Figure 5-3 Log Service Administration for the Service Instance



Log File: /var/appserver/domains/domain1/server1/logs/server.log

Default Log Level: INFO

Log stdout content to event log: ☑

Log stderr content to event log: ☑

Echo to stderr: ☑

Log virtual server id: □

Write to system log: □

You can configure the following attributes for the log service element through the Administration interface, as shown in the figure "Log Service Administration for the Service Instance" on page 113.

- Log File
- Default Log Level
- Log Standard Out content to event log
- Log Standard Error content to event log
- Echo to Standard Error
- Create Console
- Log Virtual Server ID
- Write to System Log

The Log Service link is accessible from the expanded tree hierarchy for the server instance in the left pane of the Administration interface. The following table describes each of the attributes that can be configured, along with the default and range of values allowed.

Table 5-5 Log Service Attributes

Attribute	Default	Description
`file`	`server.log`[1]	(optional) Overrides the name or location of the server log. The file and directory in which the server log is kept must be writable by whatever user account used to run the server.
`level`	INFO	(optional) Controls the default type of messages logged by other elements to the server log. Allowed values are as follows, from highest to lowest: `FINEST`, `FINER`, `FINE`, `CONFIG`, `INFO`, `WARNING`, `SEVERE`, `ALERT`, `FATAL`. Each value logs all messages for all lower values; for example, `FINEST` logs all messages, and `FATAL` logs only `FATAL` messages. The default value is `INFO`, which logs all `INFO`, `WARNING`, `SEVERE`, `ALERT`, and `FATAL` messages.
`log-stdout`	True	(optional) If `true`, redirects `stdout` output to the server log. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.

Table 5-5 Log Service Attributes

Attribute	Default	Description
`log-stderr`	True	(optional) If `true`, redirects `stderr` output to the server log. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`echo-log-messages-to-stderr`	True	(optional) If `true`, sends log messages to `stderr` in addition to the server log. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`create-console`	False	(optional) If `true`, creates a console window on Windows operating system for `stderr` output. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`log-virtual-server-id`	False	(optional) If `true`, virtual server IDs are displayed in the virtual server logs. These are useful if multiple `virtual-server` elements share the same log file.
`use-system-log`	False	If true, uses the UNIX `syslog` service or Windows Event Logging to produce and manage logs.

1. In the directory specified by the log-root attribute of the server element.

Configuring Logging for Application Server Components and Subsystems

This section describes how to enable logging and select a log level for Sun ONE Application Server components and subsystems. Note that the Java Transaction Service component has more than one log file. Since most of the components and subsystems are handled in the same way with respect to configuring a log level, the procedure to select a log level is documented one time only, for the indicated grouping of components and subsystems.

The following components and subsystems can utilize selective logging of server messages. You can become familiar with the components and subsystems from the references to other topics within this guide as indicated.

- ORB - Configuring Support for Corba-based Clients
- Web Container - Configuring J2EE Services
- EJB Container - Configuring J2EE Services
- MDB Container - Configuring J2EE Services (within the EJB container)
- Java Transaction Service - Configuring J2EE Services

- JMS Service - Java Message Service
- Virtual Server - Using Virtual Servers

To Specify a Log Level

To specify a log level for ORB, Web Container, EJB Container, MDB Container (within the EJB Container), Java Transaction Service, and JMS Service, perform the following procedure:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance to display the components and subsystems you wish to edit.
2. Click the link for the desired component or subsystem.
3. In the right page of the Administration interface, select one of the following log level parameters, from the Log Level drop-down list. The log levels are described in "About Log Levels" on page 97.

To Specify a Log File: (Virtual Server)

To specify a log file, perform the following procedure:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance to display the HTTP Server subsystem.
2. Click the HTTP Server link.
3. Click the Virtual Server link.
4. Click the desired server instance link.
5. In the right page of the Administration interface, under the General tab, enter the desired directory path and file name for the Log File field.

To Specify a Transaction Log Location: (Java Transaction Service)

To specify a transaction log location, perform the following procedure:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance to display the Transaction Service subsystem.
2. Click the Transaction Service link.
3. In the right page of the Administration interface, under the Advanced field group, enter the desired directory path and file name for the Transaction Log Location field.

Configuring the Directives for Error Logging

Sun ONE Application Server includes error logging directives for the `init.conf` file. The following directives are included:

- **Error Log Date Format.** The `ErrorLogDateFormat` directive specifies the date format that the server logs use.
- **Log Flush Interval**. The `LogFlushInterval` determines the maximum time interval, in seconds, before which the access log is flushed from the memory into the `access.log` file.
- **Pid Log.** `PidLog` specifies a file in which to record the process ID (pid) of the base server process. Some of the server support programs assume that this log is in the server root, in `logs/pid`.

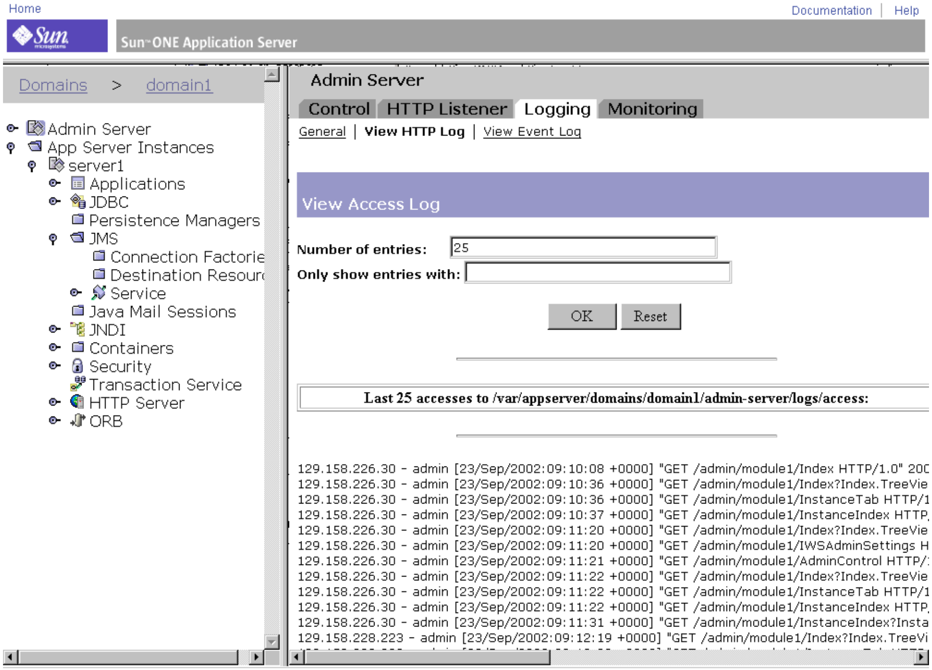
All the directives for `init.conf` are fully described in the *Sun ONE Application Server Configuration File Reference*.

Viewing the Access Log File

You can view both the Administrator server's and the Sun ONE Application Server instance's http log files.

To view the Administration Server's http log, select the Admin Server from the left pane in the Administration interface, and then choose the Logging tab from the right page. The View HTTP Access Log link is displayed. Select this link to view the configured access log. An example of the displayed log is shown in the figure "Admin Server View HTTP Access Log" on page 118.

Figure 5-4 Admin Server View HTTP Access Log



Number of entries: 25

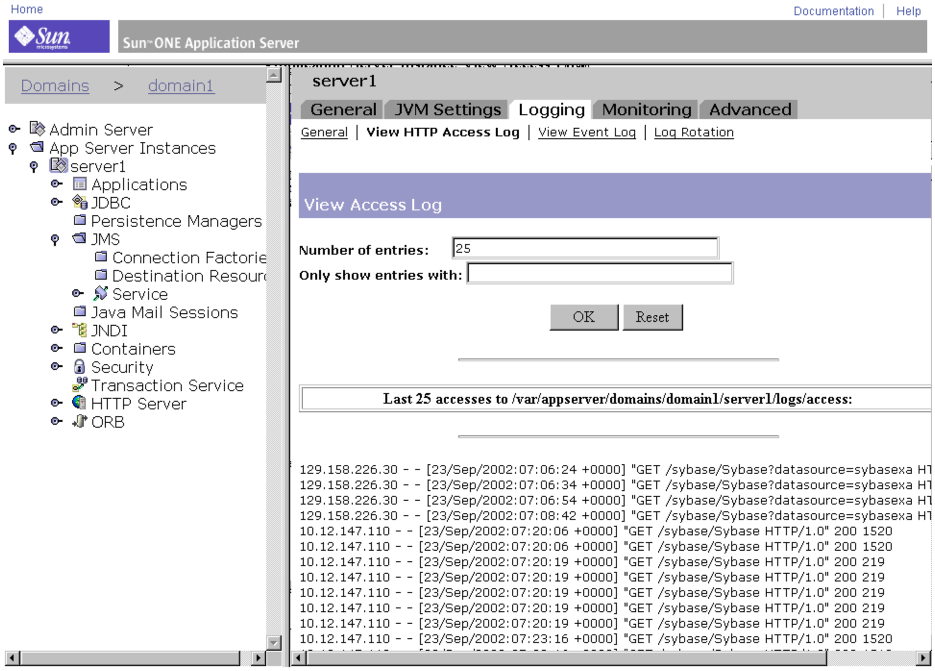
Only show entries with:

Last 25 accesses to /var/appserver/domains/domain1/admin-server/logs/access:

```
129.158.226.30 - admin [23/Sep/2002:09:10:08 +0000] "GET /admin/module1/Index HTTP/1.0" 200
129.158.226.30 - admin [23/Sep/2002:09:10:36 +0000] "GET /admin/module1/Index?Index.TreeVie
129.158.226.30 - admin [23/Sep/2002:09:10:36 +0000] "GET /admin/module1/InstanceTab HTTP/1
129.158.226.30 - admin [23/Sep/2002:09:10:37 +0000] "GET /admin/module1/InstanceIndex HTTP
129.158.226.30 - admin [23/Sep/2002:09:11:20 +0000] "GET /admin/module1/Index?Index.TreeVie
129.158.226.30 - admin [23/Sep/2002:09:11:20 +0000] "GET /admin/module1/IWSAdminSettings H
129.158.226.30 - admin [23/Sep/2002:09:11:21 +0000] "GET /admin/module1/AdminControl HTTP/
129.158.226.30 - admin [23/Sep/2002:09:11:22 +0000] "GET /admin/module1/Index?Index.TreeVie
129.158.226.30 - admin [23/Sep/2002:09:11:22 +0000] "GET /admin/module1/InstanceTab HTTP/1
129.158.226.30 - admin [23/Sep/2002:09:11:22 +0000] "GET /admin/module1/InstanceIndex HTTP
129.158.226.30 - admin [23/Sep/2002:09:11:31 +0000] "GET /admin/module1/InstanceIndex?Insta
129.158.228.223 - admin [23/Sep/2002:09:12:19 +0000] "GET /admin/module1/Index?Index.TreeVi
```

To view an access log for the application server instance, click the server instance desired from the left pane of the Administration interface. Click on the Logging tab from the right pane. Click the View Access Log link to display the configured active access log for that server instance. An example is shown in the figure "Application Server Instance View Access Log" on page 119.

Figure 5-5 Application Server Instance View Access Log



Number of entries: 25

Only show entries with:

Last 25 accesses to /var/appserver/domains/domain1/server1/logs/access:

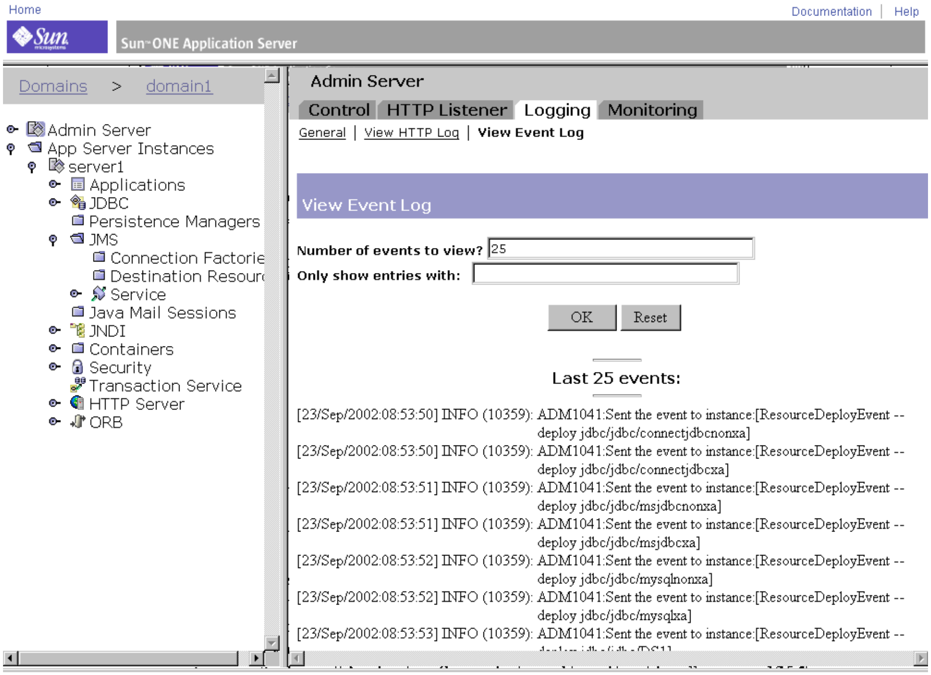
```
129.158.226.30 - - [23/Sep/2002:07:06:24 +0000] "GET /sybase/Sybase?datasource=sybasexa HT
129.158.226.30 - - [23/Sep/2002:07:06:34 +0000] "GET /sybase/Sybase?datasource=sybasexa HT
129.158.226.30 - - [23/Sep/2002:07:06:54 +0000] "GET /sybase/Sybase?datasource=sybasexa HT
129.158.226.30 - - [23/Sep/2002:07:08:42 +0000] "GET /sybase/Sybase?datasource=sybasexa HT
10.12.147.110 - - [23/Sep/2002:07:20:06 +0000] "GET /sybase/Sybase HTTP/1.0" 200 1520
10.12.147.110 - - [23/Sep/2002:07:20:06 +0000] "GET /sybase/Sybase HTTP/1.0" 200 1520
10.12.147.110 - - [23/Sep/2002:07:20:19 +0000] "GET /sybase/Sybase HTTP/1.0" 200 219
10.12.147.110 - - [23/Sep/2002:07:20:19 +0000] "GET /sybase/Sybase HTTP/1.0" 200 219
10.12.147.110 - - [23/Sep/2002:07:20:19 +0000] "GET /sybase/Sybase HTTP/1.0" 200 219
10.12.147.110 - - [23/Sep/2002:07:20:19 +0000] "GET /sybase/Sybase HTTP/1.0" 200 219
10.12.147.110 - - [23/Sep/2002:07:20:19 +0000] "GET /sybase/Sybase HTTP/1.0" 200 219
10.12.147.110 - - [23/Sep/2002:07:23:16 +0000] "GET /sybase/Sybase HTTP/1.0" 200 1520
```

Viewing the Event Log File

You can view both the Administrator server's and the Sun ONE Application Server instance's active event log files.

To view the Administrator server's event log, select the Admin Server from the left pane, and then choose the Logging tab from the right page. The View Event Log link will be displayed. Select this link to view the configured event log. An example of the displayed log is shown in the figure "Admin Server View Event Log" on page 120.

Figure 5-6 Admin Server View Event Log



Number of events to view? 25

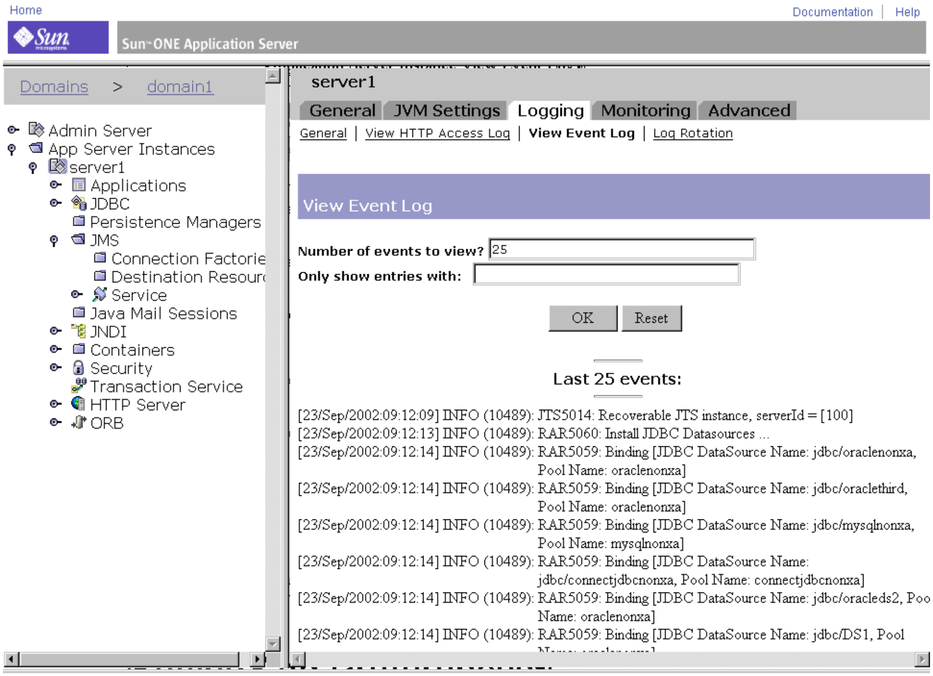
Only show entries with:

Last 25 events:

```
[23/Sep/2002:08:53:50] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/connectjdbcnonxa]
[23/Sep/2002:08:53:50] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/connectjdbcxa]
[23/Sep/2002:08:53:51] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/msjdbcnonxa]
[23/Sep/2002:08:53:51] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/msjdbcxa]
[23/Sep/2002:08:53:52] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/mysqlnonxa]
[23/Sep/2002:08:53:52] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
                                     deploy jdbc/jdbc/mysqlxa]
[23/Sep/2002:08:53:53] INFO (10359): ADM1041:Sent the event to instance:[ResourceDeployEvent --
```

To view the event log for the application server instance, click the server instance desired from the left pane of the Administration interface, and then choose the Logging tab from the right pane. The View Event Log link will be displayed. Select this link to view the configured event log. An example of the displayed log is shown in the figure "Application Server Instance View Event Log" on page 121.

Figure 5-7 Application Server Instance View Event Log



Number of events to view? 25

Only show entries with:

Last 25 events:

```
[23/Sep/2002:09:12:09] INFO (10489): JTS5014: Recoverable JTS instance, serverId = [100]
[23/Sep/2002:09:12:13] INFO (10489): RAR5060: Install JDBC Datasources ...
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name: jdbc/oraclenonxa,
                                     Pool Name: oraclenonxa]
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name: jdbc/oraclethird,
                                     Pool Name: oraclenonxa]
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name: jdbc/mysqlnonxa,
                                     Pool Name: mysqlnonxa]
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name:
                                     jdbc/connectjdbcnonxa, Pool Name: connectjdbcnonxa]
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name: jdbc/oracleds2, Poo
                                     Name: oraclenonxa]
[23/Sep/2002:09:12:14] INFO (10489): RAR5059: Binding [JDBC DataSource Name: jdbc/DS1, Pool
```

Setting Log Preferences

During installation, an access log file named access is created for the server. You can customize access logging for any resource by specifying whether to log accesses, what format to use for logging, and whether the server should spend time looking up the domain names of clients when they access a resource.

To use one log file for multiple virtual servers, `LogVsId` should be turned on in the `server.xml` file for the event log. See the *Sun ONE Application Server Configuration File Reference* for details. Alternatively, `LogVsID` can be turned on in the Admin Server Logging tab of the Administration interface.

Follow the steps below to enable the Log virtual server ID from the Administration interface. The changes are effected following restart of the Administration server.

1. Click the Admin Server from the left pane of the Administration interface.

2. Click the Logging tab on the right page.

3. Click within the check box for Log virtual server ID.

4. Click the Save button to apply the changes to the Sun ONE Application Server.

This setting requires a restart of the Sun ONE Application Server for the change to take effect.

Running the Log Analyzer

The `flexanlg` is a Log Analyzer tool used for log file reporting. The Log Analyzer can be used only when logging is directed to a file other than `syslog`.

Use the log analyzer to generate statistics about your default server, such as a summary of activity, most commonly accessed URLs, times during the day when the server is accessed most frequently, and so on. The log analyzer cannot generate statistics for virtual servers other than the default server. However, statistics can be viewed for each virtual server as described in "Viewing the Access Log File" on page 117.

NOTE	Before running the log analyzer, you must rotate the server logs. For more information, see "Log File Management" on page 104.

You can run the log analyzer command from the command-line, by running the tool `flexanlg`, which is in the directory *install_dir*`/bin/flexanlg`.

To run `flexanlg`, type the following command and options at the command prompt:

```
flexanlg [ -P ] [-n name] [-x] [-r] [-p order] [-i file]* [ -m
metafile ]* [ o file][ c opts] [-t opts] [-l opts] [-h help]
```

Command Options (Options marked with * can be repeated).

-i *filename*

```
input log file(s)
```

```
-P
```

```
proxy log format
```

-n *servername*

```
the name of the server
```

```
-x
```

```
output in HTML

-r

resolve IP addresses to hostnames

-p [ c, t, l ]

output order; default order is counts, time statistics, and lists

-m filename

meta file(s)

-o filename

output log file; default is stdout

-c [ h, n, r, f, e, u, o, k, c, z ]

count these items; default is: h, n, r, e , u, o, k, c

   h: total hits

   n: 304 Not Modified status codes (use local copy)

   r: 302 Found status codes (redirects)

   f: 404 Not Found status codes (Document Not Found)

   e: 500 Server Error status codes (Misconfiguration)

   u: total unique URLs

   o: total unique hosts

   k: total kilobytes transferred

   c: total kilobytes saved by caches

   z: Do not count any items

-t [ sx, mx, hx, xx, z ]

find general statistics; default is: s5m5h24x10

   s (number): Find top (number) seconds of log

   m (number): Find top (number) minutes of log

   h (number): Find top (number) hours of log

   u (number): Find top (number) users of log

   a (number): Find top (number) user agents of log
   r (number): Find top (number) referers of log

   x (number): Find top (number) for miscellaneous keywords

   z: Do not find any general statistics

-l [ cx, hx ]

Make a list of the specified suboption; default is: c+3h5

   c (x, +x): most commonly accessed URLs

   x: only list x entries

   +x: only list if accessed more than x times

   h (x, +x): hosts or IP addresses most often accessing your server

   x: only list x entries

   +x: only list if accessed more than x times

   z: Do not make any lists

EXAMPLE: Using the flexanlg command
```

flexanlg -i /var/opt/SUNQappserver7/domains/domain1/server1/logs/access

NOTE Before running the log analyzer, you should archive the server logs.

Viewing Events (Windows 2000 Pro)

In addition to logging errors to the `server.log` file, Sun ONE Application Server logs severe system errors to the Event Viewer. The Event Viewer lets you monitor events on your system. Use the Event Viewer to see errors resulting from fundamental configuration problems, which can occur before the error log is opened.

To use the Event Viewer, perform the following steps:

1. From the Start menu, select Programs and then Administrative Tools. Choose Event Viewer in the Administrative Tools program group.

2. Choose Application from the Log menu.

   The Application log appears in the EventViewer. Errors from Sun ONE Application Server has a source label of `https-`*`serverid`*.

3. Choose Find from the View menu to search for one of these labels in the log. Choose Refresh from the View menu to see updated log entries.

For more information about the Event Viewer, consult your system documentation.

Chapter 6

Monitoring the Sun ONE Application Server

This chapter contains information about the monitoring and Simple Network Management Protocol (SNMP) features and functions available in the Sun ONE Application Server.

This chapter includes the following sections:

- About Monitoring the Sun ONE Application Server
- Extracting Monitoring Data Using the CLI
- Administering the Transaction Service Using the CLI
- Using HTTP Quality of Service
- About SNMP
- Setting Up SNMP
- Enabling and Starting the SNMP Master Agent

About Monitoring the Sun ONE Application Server

You can monitor the Sun ONE Application Server by collecting activity statistics from strategic data points on your system. The statistics show you how many requests your server is handling and how well it is handling those requests. You can view some statistics for individual virtual servers and others for the entire application server instance. Either the `asadmin` utility or SNMP can be used to monitor the Sun ONE Application Server.

The following topics are addressed in this section:

- Statistics
- SNMP
- HTTP Server Monitoring
- Application Components and Subsystems Monitoring
- Quality of Service (QOS)

Statistics

Statistics collection is always enabled for most of the Sun ONE Application Server application components and subsystems, including the HTTP server; no enabling functions are required. However, there are some statistics that are collected only when monitoring is explicitly enabled on that subsystem, or only when relevant functionality is enabled. These statistics include the following data points:

- Statistics for EJB methods
- Active transactions
- Connections (only if Quality of Service is enabled)
- DNS (only if DNS Cache is enabled)

Monitoring can be enabled for application subsystems or components from the Administration interface, as described in "Application Components and Subsystems Monitoring" on page 129.

If the server monitor reports that the server is handling a large number of requests, you may need to adjust the server configuration or the system's network kernel. For more information on adjusting your server's configuration, see the *Sun ONE Application Server Performance Tuning and Sizing Guide.*

SNMP

The Sun ONE Application Server provides network management information through its information gathering tools using Simple Network Management Protocol (SNMP), a protocol used to exchange management and monitoring information across a network. Using SNMP, programs called *agents* monitor various devices on the network (hubs, routers, bridges, and so on). Another program collects the data from the agents. The database created by the monitoring operations is called a *management information base* (MIB).This data is used to check if all devices on the network are operating properly.

While only the HTTP server can be monitored using SNMP; all components and systems can be monitored using the command-line interface (CLI)

For more information on SNMP, see "About SNMP" on page 160 and "Setting Up SNMP" on page 168.

HTTP Server Monitoring

HTTP server monitoring is enabled by default, which means it doesn't need to be specifically turned on. HTTP server monitoring is based on an XML file and is accessed using the `asadmin` command as a set of three monitorable attributes. The elements, subelements, and the attributes of this XML file are described in "Monitorable HTTP Server Elements" on page 144 and "Monitorable HTTP Server Attributes" on page 145.

NOTE Only HTTP server statistics are available using SNMP. Statistics for all the subsystems of the Sun ONE Application Server, including the HTTP server, are available using the command-line interface.

For more information about using `asadmin`, see "Using the Command Line Interface" on page 413.

Application Components and Subsystems Monitoring

Some of the subsystems or components within the Sun ONE Application Server do not need to have monitoring enabled because the relevant statistics are always collected. For example, monitoring for application components such as containers can be enabled or disabled. When monitoring is enabled, in addition to the statistics that are always collected, additional statistics on all EJB methods are also collected. Monitoring for JDBC connection pools is always enabled. A connection pool is initialized upon first access and relevant statistics can be monitored anytime after that.

For a full list of monitorable data points, see "Monitorable Attribute Names" on page 139.

You can enable monitoring for selected application components and subsystems from the Administration interface or from the command-line interface (CLI). For example, to enable monitoring from the CLI for the EJB container, type the following command from a terminal window:

```
set server1.ejb-container.monitoringEnabled=true
reconfig server1
```

where `server1` is the instance name.

The equivalent functionality can be accessed on the Administration interface under the Containers node.

The following topics are addressed in this section:

- Monitoring for Container Subsystems
- Monitoring for the ORB Service
- Monitoring for the Transaction Service

Monitoring for Container Subsystems

In the case of the EJB container, when monitoring is enabled, the statistics related to the methods for all entity beans, stateful session beans, and stateless session beans are collected. These statistics include:

- Total number of errors
- Total number of calls
- Total number of successes
- Execution time, in milliseconds (for last invocation of the method)

All other statistics for container subsystems are always collected. Some of the monitored data points include statistics relative to:

- Initial, minimum, and maximum stateless beans in pool
- Minimum and preferred number of stateful and entity beans in cache
- Minimum and preferred number of stateless session beans in cache
- Number of beans created and destroyed
- Other related statistics

Monitoring for the ORB Service

For the ORB service, monitored data points include statistics gathered for the ORB connection and the ORB thread pool. ORB statistics are always collected and, therefore, it is not necessary to enable monitoring for the ORB service.

Monitoring for the Transaction Service

For the Java Transaction Service (JTS) service, monitored data points include:

- Total completed transactions
- Total rolled back transactions
- Total inflight transactions
- List of inflight transactions

Refer to "Administering the Transaction Service Using the CLI" on page 153 for further information.

Quality of Service (QOS)

Quality of Service refers to the performance limits you set for a server instance virtual server class or virtual server. For example, if you are an Internet Service Provider (ISP), you might want to charge different fees for virtual servers depending on how much bandwidth is provided. You can limit two areas: the amount of bandwidth and the number of connections.

The Quality of Service information provided by the Sun ONE Application Server is used to determine server efficiency during runtime with respect to:

- Start-up time
- Server traffic and effects of traffic upon bandwidth
- Analysis of live versus static data
- Other data elements

For more information, see "Administering the Transaction Service Using the CLI" on page 153.

Extracting Monitoring Data Using the CLI

With the `asadmin` command, you can extract monitored data through the command-line interface (CLI) using the `list` and `get` commands.

NOTE The `set` command is only used to set monitoring for the transaction service as described in "Administering the Transaction Service Using the CLI" on page 153.

This section addresses the following topics:

- The list --monitor Command
- The get --monitor Command
- CLI Name Mapping
- HTTP Server Monitorable Objects

The list --monitor Command

The `list` command provides information about the application components and subsystems currently being monitored for the specified server instance name. Using this command, you can see the monitorable components and sub-components for a server instance.

Example

```
asadmin> list --monitor server1
```

returns the following list of application components and subsystems that have monitoring enabled:

```
iiop-service
transaction-service
application.converter
application.myApp
http-server
```

You can also list applications that are currently monitored in the specified server instance. This can be useful when particular monitoring statistics are sought from an application using the `get` command.

Example

```
asadmin> list --monitor server1.application
```

returns:

```
converter
myApp
```

For a more comprehensive example, see “Petstore Example” on page 135.

The get --monitor Command

This command retrieves the following monitored information:

- All attribute(s) monitored within a component or subsystem
- Specific attribute monitored within a component or subsystem

When an attribute is requested that does not exist for a particular component or subsystem, an error is returned. Similarly, when a specific attribute is requested that is not active for a component or subsystem, an error is returned.

Refer to “CLI Name Mapping” on page 134 for more information on the use of the `get` command.

Example 1

Attempt to get all attributes from a subsystem for a specific attribute:

```
asadmin> get --monitor server1.iiop-service.orb.system.orb-connection.*
```

```
total-inbound-connections=1
total-outbound-connections=1
```

Example 2

Attempt to get all attributes from a J2EE application:

```
asadmin> get --monitor server1.application.converter.*
```

```
Attribute name(s) not found
```

There are no monitorable attributes exposed at the J2EE-application level, therefore the command fails.

Example 3

Attempt to get a specific attribute from a subsystem:

```
asadmin> get --monitor server1.transaction-service.inflight-tx
```

```
Attribute name = inflight-tx Value = No active transaction found.
```

Example 4

Attempt to get an unknown attribute from within a subsystem attribute:

```
asadmin> get --monitor server1.iiop-service.orb.system.orb-connection.bad-name
```

```
Could not get the attribute
```

```
Execution failed for the command: get --monitor
server1.iiop-service.orb-connection.bad-name
```

CLI Name Mapping

The Sun ONE Application Server uses a tree structure to track monitorable objects. Every node in the tree has a name and a type. If a type is singleton, only a single node of the type can exist under any parent node. For more information on type of nodes in this tree, see "Monitorable Object Types" on page 137.

The root object in the tree is represented by the Sun ONE Application Server instance name. For example, the root monitoring object for an instance named `server1` is:

```
server1
```

All child objects are then addressed using the dot (.) character as separator. If a child node is of singleton type, then only the monitoring object type is needed to address the object, otherwise a name of the form `type.name` is needed to address the object.

For example, `http-server` is one of the valid monitorable object types and is singleton. To address a singleton child node representing the `http-server` of instance `server1`, the name is:

```
server1.http-server
```

Another example, `application`, is a valid monitorable object type and is not a singleton. To address a non-singleton child node representing the application Petstore, the name is:

```
server1.application.petstore
```

The CLI names can also address specific attributes in monitorable objects. For example, `http-server` has a monitorable attribute called `summary`. The following name addresses the summary attribute:

```
server1.http-server.summary
```

There is no fixed naming convention for attribute names exposed by monitoring objects.

You are not expected to know the valid names for CLI use. The `list` command lets you inspect available monitorable objects, while the `get` command used with a wildcard parameter allows you to inspect all available attributes on any monitorable object.

The following example illustrate some client name mapping scenarios:

Petstore Example

A user wants to inspect the number of calls made to a method in the Petstore application deployed on the Sun ONE Application Server instance named `server1`. A combination of the `list` and `get` commands is used to access desired statistics on a method.

1. Invoke the CLI in multi-mode.
2. Set some useful environment variables to avoid entering them for every command:

   ```
   asadmin>export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=admin123
   ```

   ```
   asadmin>export AS_ADMIN_HOST=localhost AS_ADMIN_PORT=4848
   ```

3. List monitorable components for instance `server1`:

   ```
   asadmin>list --monitor server1
   ```

   Output is:

   ```
   iiop-service
   transaction-service
   application.CometEJB
   application.ConverterApp
   application.petstore
   http-server
   resources
   ```

   The list of monitorable components includes `iiop-service`, `http-server`, `transaction-service`, `resources`, and all deployed (and enabled) applications.

4. List the monitorable subcomponents in the Petstore application (`-m` can be used instead of `--monitor`):

```
asadmin>list -m server1.application.petstore
```

Output is:

```
ejb-module.signon-ejb_jar
ejb-module.catalog-ejb_jar
ejb-module.uidgen-ejb_jar
ejb-module.customer-ejb_jar
ejb-module.petstore-ejb_jar
ejb-module.AsyncSenderJAR_jar
ejb-module.cart-ejb_jar
```

5. List the monitorable subcomponents in the EJB module `sigon-ejb_jar` of the Petstore application:

```
asadmin>list -m server1.application.petstore.ejb-module.signon-ejb_jar
```

Output is:

```
entity-bean.UserEJB
stateless-session-bean.SignOnEJB
```

6. List the monitorable subcomponents in the entity bean `UserEJB` for the EJB module `sigon-ejb_jar` of the Petstore application:

```
asadmin>list -m
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB
```

Output is:

```
bean-method.create0
bean-method.findByPrimaryKey1
bean-method.remove2
bean-method.getUserName3
bean-method.setPassword4
bean-method.getPassword5
bean-method.matchPassword6
bean-method.remove7
bean-method.isIdentical8
bean-method.getEJBLocalHome9
bean-method.getPrimaryKey10
bean-pool
bean-cache
```

7. List the monitorable subcomponents in the method `getUserName3` for the entity bean `UserEJB` in the EJB module `sigon-ejb_jar` of the Petstore application:

```
asadmin>list -m
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB.bean-m
ethod.getUserName3
```

Output is:

```
No monitorable entities for element
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB.bean-m
ethod.getUserName3
```

8. There are no monitorable subcomponents for methods. Get all monitorable statistics for the method `getUserName3`.

```
asadmin>get -m server1.application.petstore.ejb-module.
signon-ejb_jar.entity-bean.UserEJB.bean-method.getUserName3.*
method-name = public abstract java.lang.String
com.sun.j2ee.blueprints.signon.user.ejb.UserLocal.getUserName()
total-num-errors = 0
total-num-success = 2
execution-time-millis = 1
total-num-calls = 2
```

9. You can also get a specific statistic, such as execution time.

```
asadmin>get -m server1.application.petstore.ejb-module.
signon-ejb_jar.entity-bean.UserEJB.bean-method.getUserName3.execution-time-millis
execution-time-millis = 1
```

Monitorable Object Types

The tree of objects used for monitoring contains several nodes. A node is a specific entry in the object tree, and is identified uniquely by its type, name, and parent node. Some of the node types are singleton, meaning that only one node of the type can exist under a parent node. A name is not relevant for a singleton node.

A non-singleton type node needs a name. The Instance Name column describes the possible namespace.

The following table describes the tree structure in terms of possible parent-child relationships among various node types and the namespaces for some of the node types.

Table 6-1 Monitoring Object Types

Node Type	Single-ton?	Leaf?	Child Node Types	Instance Names

`root`	Yes	No	`http-server`
			`iiop-service`
			`resources`
			`transaction-service`
			`application`
			`standalone-ejb-module`	

`http-server`	Yes	No	`virtual-server` `process`	
`virtual-server`	Yes	Yes		
`process`	Yes	Yes		
`iiop-service`	Yes	Yes	`orb`	
`orb`	No	No	`orb-connection` `orb-thread-pool`	`system` is reserved for system ORB. All user ORBs get a name derived from TCP endpoint.
`orb-connection`	Yes	Yes		
`orb-thread-pool`	Yes	Yes		
`resources`	Yes	No	`jdbc-connection-pool`	
`jdbc-connection-pool`	No	Yes		The names are the same as those specified by the user while creating a connection pool.
`transaction-service`	Yes	Yes		
`application`	No	No	`ejb-module`	Name of the application as registered in `server.xml`.

`ejb-module`	No	No	`stateless-session-bean`
			`stateful-session-bean`
			`entity-bean`
			`message-driven-bean`
				Name of the EJB module. It is derived from the EJB JAR name.

Table 6-1 Monitoring Object Types *(Continued)*

Node Type	Single-ton?	Leaf?	Child Node Types	Instance Names

`standalone-ejb-module`	No	No	`stateless-session-bean`
			`stateful-session-bean`
			`entity-bean`
			`message-driven-bean`
				Name of the standalone EJB module as registered in `server.xml`.

`stateless-session-bean`	No	No	`bean-pool` `bean-method`	Name of the bean from the deployment descriptor.
`stateful-session-bean`	No	No	`bean-cache` `bean-method`	Name of the bean from the deployment descriptor.

`entity-bean`	No	No	`bean-cache`
			`bean-pool`
			`bean-method`
				Name of the bean from the deployment descriptor.

`message-driven-bean`	No	No	`bean-pool` `bean-method`	Name of the bean from the deployment descriptor.
`bean-pool`	Yes	Yes		
`bean-cache`	Yes	Yes		
`bean-method`	No	Yes		`onMessage` for message-driven beans, method name followed by a numeric suffix for methods in other enterprise beans. (The suffix is needed to disambiguate overloaded methods.)

Monitorable Attribute Names

It is not necessary for every monitorable object to expose monitorable attributes. Some of the objects are used just for grouping other objects. For the Sun ONE Application Server, except for an `http-server` node, only the leaf nodes of the tree have attributes. The `http-server` node type has attributes as well as child nodes. The following tables list possible monitorable attribute names for various nodes.

Table 6-2 http-server

Attribute Name	Datatype	Description
`summary`	String (Formatted)	HTTP server summary. Includes virtual servers and processes. NOTE: See section "HTTP Server Monitorable Objects" on page 144 for more information on what data is exposed in the formatted string.

Table 6-3 virtual-server

Attribute Name	Datatype	Description
`<vs-id>`	String (Formatted)	Virtual server information. For every application server instance there can be one or more virtual servers. The list of virtual server IDs can be obtained from the summary attribute of `http-server`. You can look up statistics for a specific virtual server using the `get` command parameter of the form `server1.http-server.virtual-server.<vs-id>`. You can look up statistics for all virtual servers using the `get` command parameter of the form `server1.http-server.virtual-server.*`. NOTE: See section "HTTP Server Monitorable Objects" on page 144 for more information on what data is exposed in the formatted string.

Table 6-4 process

Attribute Name	Datatype	Description
`<pid>`	String (Formatted)	Process information. For every application server instance there is one process. The process ID can be obtained from the summary attribute of `http-server`. Statistics for a process can be obtained using the parameter of the form `server1.http-server.process.<pid>` to the `get` command. NOTE: See section "HTTP Server Monitorable Objects" on page 144 for more information on what data is exposed in the formatted string.

Table 6-5 orb-connection

Attribute Name	Datatype	Description
`total-inbound-connections`	Integer	Total inbound connections to ORB.
`total-outbound-connections`	Integer	Total outbound connections from ORB.

Table 6-6 orb-thread-pool

Attribute Name	Datatype	Description
`thread-pool-size`	Integer	Total number of threads in ORB thread pool.
`waiting-thread-count`	Integer	Number of thread pool threads waiting for work to arrive.

Table 6-7 jdbc-connection-pool

Attribute Name	Datatype	Description
`total-threads-waiting`	Integer	Total threads waiting for JDBC connection.
`total-outbound-connections`	Integer	Total JDBC connection validation failures.
`total-connections-timed-out`	Integer	Total connection requests timed out.

Table 6-8 transaction-service

Attribute Name	Datatype	Description
`total-tx-completed`	Integer	Total completed transactions.
`total-tx-rolled-back`	Integer	Total rolled back transactions.
`total-tx-inflight`	Integer	Total inflight (live) transactions.
`isFrozen`	String	Is transaction system frozen (true or false)?
`inflight-tx`	String (Formatted)	List of inflight transactions.

Table 6-9 bean-pool

Attribute Name	Datatype	Description
`max-pool-size`	Integer	The maximum number of bean instances in the pool.
`steady-pool-size`	Integer	The number of bean instances normally maintained in the pool. When a pool is first created, it will be populated with a size equal to `steady-pool-size`. When an instance is removed from the pool, it is replenished asynchronously, so that the pool size is at, or above, `steady-pool-size`.
`pool-resize-quantity`	Integer	The increment by which pool grows up to `max-pool-size` or shrinks to `steady-pool-size`
`idle-timeout-in-seconds`	Integer	Defines the rate at which the pool cleaning thread is executed. Checks if the current size is greater than the steady pool size and removes `pool-resize-quantity` elements. If the current size is less than `steady-pool-size`, it is increased by `pool-resize-quantity`, with a ceiling of `min` (`current-pool-size+pool` + `resize-quantity`, `max-pool-size`). Only objects that have not been accessed for more than `pool-idle-timeout-in-seconds` are candidates for removal
`num-beans-in-pool`	Integer	Number of beans available in pool.
`num-threads-waiting`	Integer	Number of threads waiting for free beans.
`total-beans-created`	Integer	Number of beans created so far.
`total-beans-destroyed`	Integer	Number of beans destroyed so far.
`jms-max-messages-load`	Integer	The maximum number of messages to load into a JMS session at one time for a message-driven bean to serve. Default is 1. Applies only to pools for message driven beans.

Table 6-10 bean-cache

Attribute Name	Datatype	Description

`cache-resize-quantity`
`(resize-quantity)`
	Integer	The quantity by which the cache size is reduced when the number of beans in cache equals `max-cache-size` (that is, when cache overflow occurs.)

`cache-misses`	Integer	The number of times a user request did not find a bean in the cache.

Table 6-10 bean-cache *(Continued)*

Attribute Name	Datatype	Description
`idle-timeout-in-seconds`	Integer	Rate at which the cache cleaner thread is scheduled. This cleaner thread examines all beans in the cache and passivates those beans that are not accessed for `cache-idle-timeout-in-seconds`.
`cache-hits`	Integer	The number of times a user request found an entry in the cache.
`total-beans-in-cache`	Integer	The number of beans in the cache. This is the current size of the cache.
`max-beans-in-cache`	Integer	Maximum number of beans that can be held in the cache beyond which cache overflow occurs.
`num-passivations`	Integer	Number of passivations. Applies only to stateful session beans.
`num-passivation-errors`	Integer	Number of errors during passivation. Applies only to stateful session beans.
`num-expired-sessions-removed`	Integer	Number of expired sessions removed by the cleanup thread. Applies only to stateful session beans.
`num-passivation-success`	Integer	Number of times passivation completed successfully. Applies only to stateful session beans.

Table 6-11 bean-method

Attribute Name	Datatype	Description
`method-name`	String	Fully qualified name of the method.
`total-num-calls`	Integer	Number of times the method has been invoked. This is collected for stateless and stateful session beans and entity beans if monitoring enabled is true for EJB container, and for message-driven beans if monitoring is enabled for the message-driven bean container.
`total-num-errors`	Integer	Number of times the method execution resulted in an exception. This is collected for stateless and stateful session beans and entity beans if monitoring is enabled under EJB settings, and for message-driven beans if monitoring is enabled under MDB settings.

Table 6-11 bean-method *(Continued)*

Attribute Name	Datatype	Description
`total-num-success`	Integer	Number of times the method successfully executed. This is collected for stateless and stateful session beans and entity beans if monitoring enabled is true for EJB container, and for message-driven beans if monitoring is enabled for the container.
`execution-time-millis`	Long	Time spent executing the method for the last successful run of this method. This is collected for stateless and stateful session beans and entity beans if monitoring is enabled on the EJB container and for message-driven beans if monitoring is enabled on the container.

HTTP Server Monitorable Objects

The HTTP server monitorable attribute name `summary` prints the attribute values of the `Server` element and a summary of its subelements, including the number of each subelement and attribute values for each subelement. The HTTP server `virtual-server` attribute prints the attribute values of the `VirtualServer` element and the details of each of its subelements.The `process` attribute prints the attribute values of the `Process` element and the details of each of its subelements.

To enable NSAPI performance profiling and obtain statistics on the `Profile` and `ProfileBucket` elements, see the *Sun ONE Application Server Developer's Guide to NSAPI*.

For information on how to use the monitoring statistics for performance tuning, see the *Sun ONE Application Server Performance and Tuning Guide*.

Monitorable HTTP Server Elements

The following table lists the HTTP server monitorable elements.

Table 6-12 Monitorable HTTP Server Elements

Element Name	Subelements	Description
`Server`	ConnectionQueue ThreadPool Profile Process VirtualServer	A server instance.

Table 6-12 Monitorable HTTP Server Elements *(Continued)*

Element Name	Subelements	Description
ConnectionQueue	None	The queue in which requests are held prior to being serviced. There is only one connection queue in Sun ONE Application Server 7.
ThreadPool	None	A thread pool, as defined in the init.conf file.
Profile	None	An NSAPI performance profile bucket, as defined in the init.conf file.
Process	ConnectionQueueBucket ThreadPoolBucket DnsBucket DnsBucket KeepaliveBucket CacheBucket Thread	A single server process within a server instance.
ConnectionQueueBucket	None	Tracks statistics pertaining to a specific ConnectionQueue.
ThreadPoolBucket	ThreadPoolBucket	Tracks statistics pertaining to a specific ThreadPool.
DnsBucket	None	Tracks DNS statistics.
KeepaliveBucket	None	Tracks keepalive (persistent connection) statistics.
CacheBucket	None	Tracks file cache (NSFC) statistics.
Thread	RequestBucket ProfileBucket	Describes a request processing thread.
VirtualServer	RequestBucket ProfileBucket	Describes a virtual server.
RequestBucket	None	Tracks request-related statistics.
ProfileBucket		Tracks statistics pertaining to a Profile element.

Monitorable HTTP Server Attributes

The following tables list the HTTP server monitorable attributes.

Table 6-13 Server

Attribute Name	Values	Description
`Id`		The server instance ID (for example, `server1`).
`VersionServer`		A string that contains the Sun ONE Application Server version.
`TimeStarted`	GMT	The time this server instance was started.
`SecondsRunning`		The number of seconds since this server instance started.
`TicksPerSecond`		The number of ticks in a second. This value is system-dependent.
`MaxProcs`		The maximum number of processes.
`MaxThreads`		The maximum number of processing threads.
`MaxVirtualServers`		The maximum number of virtual servers tracked.
`FlagProfilingEnabled`	0 (off), 1 (on)	Indicates whether NSAPI performance profiling is enabled (on).
`FlagVirtualServerOverflow`	0 (no), 1 (yes)	Indicates whether more than `MaxVirtualServers` are configured (yes). If this attribute is set to 1, statistics are not being tracked for all virtual servers.
`LoadMinuteAverage`		Average load in 1 minute.
`Load5MinuteAverage`		Average load in 5 minutes.
`Load15MinuteAverage`		Average load in 15 minutes.
`RateBytesTransmitted`	bytes per second	The rate at which data is transmitted over some server-defined interval, or 0 if this information is not available.
`RateBytesReceived`	bytes per second	The rate at which data is received over some server-defined interval, or 0 if this information is not available.

Table 6-14 ConnectionQueue

Attribute Name	Values	Description
`Id`		The connection queue ID.

Table 6-15 ThreadPool

Attribute Name	Values	Description
`Id`		The thread pool ID.
`Name`		The symbolic name of the thread pool.

Table 6-16 Profile

Attribute Name	Values	Description
`Id`		The NSAPI performance profile bucket ID.
`Name`		The symbolic name of the NSAPI performance profile bucket.
`Description`		The description of the NSAPI performance profile bucket.

Table 6-17 Process

Attribute Name	Values	Description
`Pid`		The operating system process identifier that uniquely identifies this process.
`Mode`	unknown active	Displays active when this process is active.
`TimeStarted`	GMT	The time this process was started.
`CountConfigurations`		The number of times a configuration has been loaded, or 0 if this information is not available.
`SizeVirtual`	kilobytes	The size of virtual memory used by this process.
`SizeResident`	kilobytes	The size of the resident memory used by this process.
`FractionSystemMemoryUsage`		Fraction of system memory used by this process.

Table 6-18 ConnectionQueueBucket

Attribute Name	Values	Description
`ConnectionQueue`		The ID of a `ConnectionQueue` element.
`CountTotalConnection`		The total number of new connections that have been accepted.
`CountQueued`		The number of connections currently enqueued.
`PeakQueued`		The largest number of connections that have been in the queue simultaneously.
`MaxQueued`		The maximum number of connections that can be in the queue.
`CountOverflow`		The number of times the queue has been too full to accommodate a connection.
`CountTotalQueued`		The total number of connections that have been queued. A given connection may be queued multiple times, so `CountTotalQueued` may be greater than or equal to `CountTotalConnections`.
`TicksTotalQueued`		The total number of ticks that connections have spent in the queue. A tick is a system-dependent unit of time; see `TicksPerSecond.`

Table 6-19 ThreadPoolBucket

Attribute Name	Values	Description
`Thread-pool`		The ID of a `ThreadPool` element.
`CountThreadsIdle`		The number of request processing threads currently idle.
`CountThreads`		The number of request processing threads.
`MaxThreads`		The maximum number of request processing threads that can exist concurrently.
`CountQueued`		The number of requests queued for processing by this thread pool.

Table 6-19 ThreadPoolBucket *(Continued)*

Attribute Name	Values	Description
PeakQueued		The largest number of requests that have been in the queue simultaneously.
MaxQueued		The maximum number of requests that can be in the queue.

Table 6-20 DnsBucket

Attribute Name	Values	Description
FlagCacheEnabled	0 (off), 1 (on)	Indicates whether the DNS cache is enabled (on).
CountCacheEntries		The number of DNS entries presently in the cache.
MaxCacheEntries		The maximum number of DNS entries the cache can accommodate.
CountCacheHits		The number of times a DNS cache lookup has succeeded.
CountCacheMisses		The number of times a DNS cache lookup has failed.
FlagAsyncEnabled	0 (off), 1 (on)	Indicates whether asynchronous DNS lookups are enabled (on).
CountAsyncNameLookups		The total number of asynchronous DNS name lookups performed.
CountAsyncAddrLookups		The total number of asynchronous DNS address lookups performed.
CountAsyncLookupsInProgress		The total number of asynchronous DNS lookups currently in progress.

Table 6-21 KeepaliveBucket

Attribute Name	Values	Description
CountConnections		The number of connections currently in keepalive mode.
MaxConnections		The maximum number of simultaneous keepalive connections.

Table 6-21 KeepaliveBucket *(Continued)*

Attribute Name	Values	Description
`CountHits`		The total number of times connections in keepalive mode have subsequently made a valid request.
`CountFlushes`		The number of times keepalive connections have been closed by the server.
`CountTimeouts`		The number of times keepalive connections have timed out.
`SecondsTimeouts`		The number of seconds before the server closes an idle keepalive connection.
`CountRefusals`		The number of times keepalive connections have been refused by the server.

Table 6-22 CacheBucket

Attribute Name	Values	Description
`FlagEnabled`	0 (off), 1 (on)	Indicates whether the file cache is enabled (on).
`SecondsMaxAge`	Number of seconds	The maximum age of a file cache entry.
`CountEntries`		The number of entries currently in the file cache.
`MaxEntries`		The maximum number of cache entries the file cache can accommodate simultaneously.
`CountOpenEntries`		The number of entries associated with an open file.
`MaxOpenEntries`		The maximum number of cache entries associated with an open file that the file cache can accommodate simultaneously.
`SizeHeapCache`	Number of bytes	The amount of heap used by cached file content.
`MaxHeapCacheSize`	Number of bytes	The maximum amount of heap the file cache uses for cached file content.
`SizeMmapCache`	Number of bytes	The amount of address space used by memory mapped file content.
`MaxMmapCacheSize`	Number of bytes	The maximum amount of address space that the file cache uses for memory mapped file content.
`CountHits`		The number of times a cache entry lookup has succeeded.

Table 6-22 CacheBucket *(Continued)*

Attribute Name	Values	Description
CountMisses		The number of times a cache entry lookup has failed.
CountInfoHits		The number of times a file information lookup has succeeded.
CountInfoMisses		The number of times a file information lookup has failed.
CountContentHits		The number of times a content lookup has succeeded.
CountContentMisses		The number of times a content lookup has failed.

Table 6-23 Thread

Attribute Name	Values	Description
Mode	unknown, idle, DNS, request, processing, response, updating	The thread's last known status.
TimeStarted	GMT	The time this thread was started.
ConnectionQueue		The ID of the ConnectionQueue the thread is servicing.

Table 6-24 VirtualServer

Attribute Name	Values	Description
Id		The virtual server ID.
Mode	unknown, active	Displays active when this virtual server is active.
Hosts		The software virtual server hostnames serviced by this virtual server (for example, www.foo.com foo.com foo.isp.com).
Interfaces		The interfaces (listeners) the virtual server is configured for (for example, 192.168.1.2:80 192.168.1.2:443).

Table 6-25 RequestBucket

Attribute Name	Values	Description
`CountRequests`		The number of requests serviced.
`CountBytesReceived`		The number of bytes received, or 0 if this information is not available.
`CountBytesTransmitted`		The number of bytes transmitted, or 0 if this information is not available.
`RateBytesTransmitted`	bytes per second	The rate at which data was transmitted over some server-defined interval, or 0 if this information is not available.
`MaxByteTransmissionRate`		The maximum rate at which data was transmitted over some server-defined interval, or 0 if this information is not available.
`CountOpenConnections`		The number of open connections, or 0 if this information is not available.
`MaxOpenConnections`		The maximum number of open connections, or 0 if this information is not available.
`Count2xx`		The number of 200-level responses sent.
`Count3xx`		The number of 300-level responses sent.
`Count4xx`		The number of 400-level responses sent.
`Count5xx`		The number of 500-level responses sent.
`CountOther`		The number of responses sent that were not 200, 300, 400, or 500 level.
`Count200`		The number of 200-level responses sent.
`Count302`		The number of 302-level responses sent.
`Count304`		The number of 304-level responses sent.
`Count400`		The number of 400-level responses sent.
`Count401`		The number of 401-level responses sent.
`Count403`		The number of 403-level responses sent.
`Count404`		The number of 404-level responses sent.
`Count503`		The number of 503-level responses sent.

Table 6-26 ProfileBucket

Attribute Name	Values	Description
Profile		The ID of a `Profile` element.
`Countcalls`		The number of calls to NSAPI SAFs.
CountRequests		The number of requests processed.
TicksDispatch		The number of ticks spent dispatching requests. A tick is a system-dependent unit of time; see `TicksPerSecond.`
TicksFunction		The number of ticks spent in NSAPI SAFs. A tick is a system-dependent unit of time; see `TicksPerSecond`.

Administering the Transaction Service Using the CLI

You can use the `set` command to administer the statistics you want to monitor for the JTS.

Example 1

To add a transaction to a rollback list (which results in rollback or specified transaction), issue the `set` command as follows:

```
set --monitor server1.transaction-service.rollback-list=txnid1
```

Example 2

To freeze the transaction service, issue the `set` command as follows:

```
set --monitor server1.transaction-service.freeze=true
```

The following table describes the attributes that can be monitored to gather statistics for the JTS. These attributes can be set from the command line according to the rules described in "CLI Name Mapping" on page 134.

For more information on the java transaction service, refer to Chapter 9, "Using Transaction Services."

Using HTTP Quality of Service

The following settings govern how traffic is counted and how often the bandwidth is recomputed:

- Recompute interval—Indicates how often (in milliseconds) the bandwidth is computed.
- Metric interval—The period of time for which data is used in traffic calculations.

In the Administration interface, you can enable these server or class-level settings for the server instance or for a class of virtual servers. However, you can override them for an individual virtual server.

This section includes the following topics:

- Quality of Service Example
- Configuring Quality of Service (QOS)
- Required Changes to the obj.conf File
- Known Limitations to Quality of Service

Quality of Service Example

The following example shows how the Quality of Service information is collected and computed.

- The server has metric interval of 30 seconds.
- The server starts up at a time of 0 seconds.
- At 1 second, an HTTP connection generates 5000 bytes of traffic to/from the server.
- No more connections are made after that. At 30 seconds, the total traffic for the last 30 seconds is 5000 bytes.
- At 32 seconds, the traffic sample from 1 second is discarded, since it is older than the 30 seconds of the metric interval. The total traffic for the last 30 seconds is now 0.

The recompute interval works similarly. The server's recompute interval is 100ms.

Continuing with the example, the bandwidth is recomputed periodically every 100 milliseconds. The calculation is based on the amount of traffic as well as the metric interval.

- At 0 seconds, the bandwidth is calculated for the first time. The total traffic is zero, divided by the metric interval of 30 seconds, giving a bandwidth of zero.
- At 1 second, the bandwidth is calculated for the 10th time (1000 milliseconds / 100 milliseconds). The total traffic is 5000 bytes, which is divided by 30 seconds. The bandwidth is 5000 / 30 = 166 bytes per second.
- At 30 seconds, the bandwidth is calculated for the 300th time. The total traffic is 5000 bytes, which is divided by 30 seconds. The bandwidth is 5000 / 30 = 166 bytes per second.
- At 32 seconds, the bandwidth is computed again for the 320th time. The traffic is now 0 (since the one connection that generated traffic is too old to be counted), divided by 30, giving a bandwidth of 0 bytes / second.

Configuring Quality of Service (QOS)

Quality of Service for a server instance or a class of virtual servers is configured through the Administration interface.

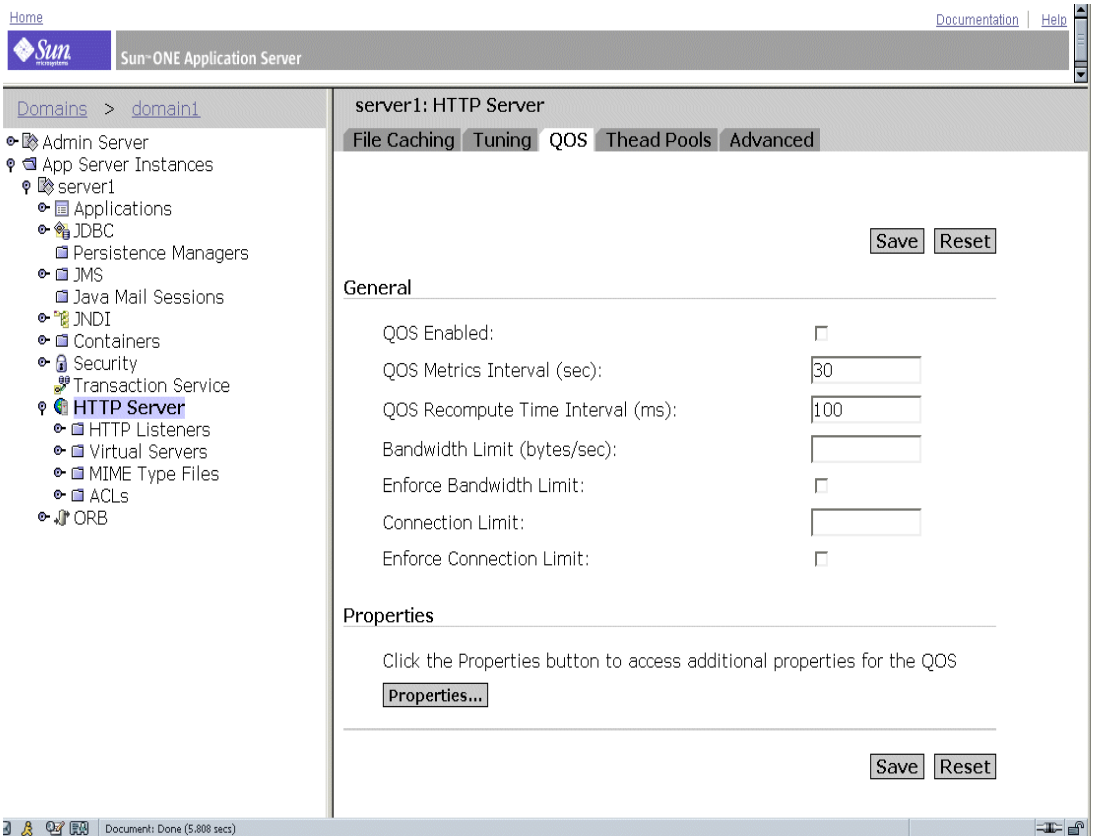
NOTE To enforce your Quality of Service settings, you must also set up Server Application Functions (SAFs) in the `obj.conf` file as described in "Required Changes to the obj.conf File" on page 158.

To configure Quality of Service, follow these steps:

1. Select the App Server Instances node in the left pane.
2. Expand the server instance node to display the HTTP Server node.
3. Click the HTTP Server node to display the QOS tab.
4. Click the QOS tab.

   A following page displays the general settings for Quality of Service, followed by a properties button.

Figure 6-1 Virtual Server Instance QOS Tab



server1: HTTP Server

General

QOS Enabled: □

QOS Metrics Interval (sec): 30

QOS Recompute Time Interval (ms): 100

Bandwidth Limit (bytes/sec):

Enforce Bandwidth Limit: □

Connection Limit:

Enforce Connection Limit: □

Properties

Click the Properties button to access additional properties for the QOS

5. To enable Quality of Service for this HTTP Server, click QOS Enable.

   Note: By default, Quality of Service is disabled. Enabling Quality of Service slightly increases server overhead.

6. Specify the QOS Metrics Interval.

   The metrics interval is the time period, in seconds, that data is sampled during server traffic calculations. The default value is 30 seconds.

   If your site commonly has large file transfers, use a large value (several minutes or more) for this field. A large file transfer might take up all the allowed bandwidth for a short metric interval, and result in connections being denied if you've enforced the maximum bandwidth setting. Since the bandwidth is averaged by the metric interval, a longer interval smooths out spikes caused by large files.

   If the bandwidth limit is much lower than available bandwidth (for example, 1 MB-per-second bandwidth limit but with a 1 GB-per-second connection to the backbone), the metric interval should be shortened.

   Note: If you have large static file transfers and a bandwidth limit that is much lower than the available bandwidth, you must decide which situation to tune for, since the problems require opposite solutions.

7. Specify the QOS Recompute Time Interval.

   The recompute time interval is the number of milliseconds between each computation of the bandwidth for all servers, classes, and virtual servers. The default is 100 milliseconds.

8. Specify the Bandwidth Limit.

   This is the maximum bandwidth limit for the server instance in bytes per second. It is interdependent with the QOS Metrics Interval to some extent.

9. Choose whether or not to enforce the maximum bandwidth setting.

   If you choose to enforce the maximum bandwidth, when the server reaches its bandwidth limit, additional connections are refused.

   If you do not enforce the maximum bandwidth, when the maximum is exceeded, the server logs a message to the event log.

10. Specify the connection limit.

    This is the number of concurrent requests processed.

11. Choose whether or not to enforce the connection limit setting.

    If you choose to enforce the maximum connections, when the server reaches its limit, additional connections are refused. If you do not enforce the maximum connections, when the maximum is exceeded, the server logs a message to the event log.

12. (Optional). If specifying additional name-value pair attributes for Quality of Service, click the Properties button.

    For a list of allowed name-value pairs for Quality of Service properties, refer to the online help.

13. Click Save to commit the changes to the server instance.

14. Access App Server Instances and your server instance in the left pane, then click Apply Changes.

Required Changes to the obj.conf File

To enforce Quality of Service, you must include directives in your `obj.conf` file to invoke following Server Application Functions (SAFs):

- AuthTrans `qos-handler`
- Error `qos-error`

In order to work properly, the `qos-handler` AuthTrans directive must be the first `AuthTrans` configured in the default object. The role of the Quality of Service handler is to examine the current statistics for the virtual server, virtual server class, and global server, and to enforce the limits by returning an error. The Sun ONE Application Server includes a built-in sample Quality of Service handler SAF, called `qos-handler`. This SAF logs when limits are reached, and returns a 503 `Server busy` error to the server so that it can be processed by the NSAPI.

The Sun ONE Application Server also includes a built-in sample error SAF called `qos-error` which returns an error page stating which limits caused the 503 error and the value of the statistic that triggered the limit.

For more information on these SAFs and how to use them, see the *Sun ONE Application Server Developer's Guide to NSAPI.*

Known Limitations to Quality of Service

When you use the Quality of Service features, keep in mind the following limitations:

- The Quality of Service features only measure the HTTP bandwidth at the application level. The HTTP bandwidth can differ from the actual TCP network bandwidth for a variety of reasons:

  - If SSL is enabled, handshakes and client certificate exchanges add to the traffic but are not measured.
  - If chunked encoding is enabled in either or both directions, the chunking layer removes the chunk headers and they are not counted in the traffic. Other headers or protocol items are counted.

- The Quality of Service features cannot accurately measure traffic from `PR_TransmitFile` calls. For basic I/O operations such as `PR_Send()/net_write` or `PR_Recv()/net_read`, the data transferred can be quickly accounted for by the bandwidth manager, since the number of bytes transferred in one system call is usually the size of a buffer and the I/O call returns quickly. This works very well to measure the instantaneous bandwidth of dynamic content applications. However, because the amount of data transferred from `PR_TransmitFile` is only known at the end of the transfer, it can't be measured before it completes.

  If the `PR_TransmitFile` is short, then the Quality of Service features will perform adequately. However, If the `PR_TransmitFile` is long, as in the case of a long file downloaded by a dialup user, the whole amount of data transferred will be counted at completion time. When the bandwidth manager recomputes bandwidth after the next recompute interval period starts, the bandwidth computed will increase significantly because of that recent large `PR_TransmitFile`. This case could cause the server to deny all requests until the next metric interval, when the bandwidth manager will "expire" the transmit file operation (since it is too old) and thus the bandwidth value will go back down. If your site commonly has very long static file downloads, you should increase the metric interval from the default 30 seconds.

- The bandwidth computed is always an approximation because it is not measured instantaneously, but is recomputed at regular intervals and over a certain period. For example, if the metric interval is the default 30 seconds and the server is idle for 29 seconds, then the next second, a client could potentially use 30 times the bandwidth limit in one second.

- The Quality of Service bandwidth statistics are lost whenever the server is reconfigured dynamically. In addition, the Quality of Service limitations are not enforced in threads that have connections on an older, inactive configuration, because the bandwidth manager thread only computes bandwidth statistics for the active configuration. Potentially, a client that doesn't close its socket for a long time and remains active so that the server doesn't time it out would not be subject to the Quality of Service limitations after a dynamic server reconfiguration.

- The concurrent connections are computed with a different granularity for virtual servers than for virtual server classes and the global server instance. The connection counter for an individual virtual server is incremented atomically immediately after the request is parsed and routed to the virtual server. It is also decremented atomically at the end of the response processing for that request. This means that the virtual server connection statistics are always exact at any instant.

  However, the connection statistics for the virtual server class and global server instance are not updated instantly. They are updated by the bandwidth manager thread every recompute interval. The connection count for the virtual server class is the sum of the connections on all virtual servers of that class; and the global server instance connection count is the sum of connections on all virtual server classes.

  Because of the way these values are computed, the number of connections for a virtual server is always correct, and if you've enforced a limit to the number of connections, you can never have more than the limit. The virtual server class and server instance values are not quite as accurate, since they are only computed at intervals.

About SNMP

Simple Network Management Protocol (SNMP) is a protocol used to exchange management and monitoring information across a network. With SNMP, data travels between a managed device and a network management station (NMS). A *managed device* is anything that runs SNMP: hosts, routers, your HTTP server, and other servers on your network.

This section addresses the following topics:

- Network Management Station (NMS)
- Management Information Base (MIB) Objects
- SNMP Messages
- SNMP Trap Destinations
- SNMP Agent Community

Network Management Station (NMS)

The *network management station* (NMS) is a machine used to remotely manage a specific network. Usually, the NMS software will provide a graph to display collected data or use that data to make sure the server is operating within a particular tolerance.

The NMS is usually a powerful workstation with one or more network management applications installed. A network management application such as HP OpenView graphically shows information about managed devices, such as your HTTP servers. For example, it might show which servers in your enterprise are up or down, or the number and type of error messages received. When you use SNMP with the Sun ONE Application Server, this information is transferred between the NMS and the server through the use of two types of agents: the subagent and the master agent.

The subagent gathers information about the server instances running in various domains and passes the information to the master agent. There is a master agent and a subagent for every installation of the Sun ONE Application Server.

NOTE After making any SNMP configuration changes, you must click the Apply button, then restart the SNMP subagent.

The *master agent* exchanges information between the various subagents and the NMS. The master agent is installed with the Sun ONE Application Server.

You can have multiple subagents installed on a host computer, but only one master agent. For example, if you had Sun ONE Directory Server, Sun ONE Application Server, and the Sun ONE Messaging Server installed on the same host, the subagents for each of the servers would communicate with the same master agent.

The NMS either requests information from the server or changes the value of a variable store in the server’s MIB. For example:

1. The NMS sends a message to the Admin Server master agent. The message might be a request for data (a `GET` message), or an instruction to set a variable in the MIB (a `SET` message).
2. The master agent forwards the message to the appropriate subagent.
3. The subagent retrieves the data or changes the variable in the MIB.
4. The subagent reports data or status to the master agent, then the master agent forwards the message back (a `GET` message) to the NMS.

5. The NMS displays the data textually or graphically through its network management application.

Management Information Base (MIB) Objects

The Sun ONE Application Server stores variables pertaining to managing and monitoring information across a network. Variables the master agent can access are called managed objects. These objects are defined in a tree-like structure called the *management information base* (MIB). The MIB provides access to the HTTP server's network configuration, status, and statistics. Using SNMP, you can view this information from the network management workstation (NMS).

The top level of the MIB tree shows that the internet object identifier has the following subtrees:

- directory (1)
- mgmt (2)
- experimental (3)
- private (4)

The private (4) subtree contains the enterprises (1) node. Each subtree in the enterprises (1) node is assigned to an individual enterprise, which is an organization that has registered its own specific MIB extensions. An enterprise can then create product-specific subtrees under its subtree. MIBs created by companies are located under the enterprises (1) node.

Each Sun ONE Application Server subagent provides a MIB for use in SNMP communication. The server reports significant events to the NMS by sending messages or traps containing these variables. The NMS can query the server's MIB for data.

Each Sun ONE Application Server has its own MIB located at *install_dir*/`lib`

The Sun ONE Application Server's MIB is a file called `appserv.mib`. This MIB contains the definitions for various variables pertaining to network management for the Sun ONE Application Server.

The Sun ONE Application Server MIB has an object identifier of

`appserver 1 (as appserver7 OBJECT IDENTIFIER ::= {appserver 1 })` and is located in the *install_dir*/`lib` directory.

You can see administrative information about your Sun ONE Application Server and monitor the server in real time using the Sun ONE Application Server MIB. The following table lists and describes the managed objects stored in the `appserv.mib` file.

`appserv.mib` Managed Objects and Descriptions

Managed object	Description
`iwsCpuID`	CPU identifier.
`iwsCpuIdleTime`	Idle CPU time.
`iwsCpuKernelTime`	CPU kernel time.
`iwsCpuTable`	Sun ONE Application Server CPUs.
`iwsCpuUserTime`	CPU user time.
`iwsInstanceTable`	Sun ONE Application Server instances.
`iwsInstanceId`	Server instance identifier
`iwsInstanceVersion`	String, such as `SunONE-ApplicationServer-Enterprise/7 BB1-01/24/2001 17:15 (SunOS DOMESTIC)`
`iwsInstanceDescription`	Description of the server instance.
`iwsInstanceOrganization`	Organization responsible for the server instance.
`iwsInstanceContact`	Contact information for person(s) responsible for server instance.
`iwsInstanceLocation`	Where the server is located.
`iwsInstanceStatus`	Status of the server instance.
`iwsInstanceUptime`	How long the server has been running.
`iwsInstanceDeathCount`	Number of times server instance processes have gone down.
`iwsInstanceRequests`	Number of requests processed by the server instance.
`iwsInstanceInOctets`	Number of octets received by the server instance. Will show 0 if information is not available.
`iwsInstanceOutOctets`	Number of octets transmitted by the server instance. Will show 0 if information is not available.
`iwsInstanceCount2xx`	Number of 200-level (Successful) responses issued by the server instance.
`iwsInstanceCount3xx`	Number of 300-level (Redirection) responses issued by the server instance.

appserv.mib Managed Objects and Descriptions *(Continued)*

Managed object	Description
iwsInstanceCount4xx	Number of 400-level (Client Error) responses issued by the server instance.
iwsInstanceCount5xx	Number of 500-level (Server Error) responses issued by the server instance.
iwsInstanceCountOther	Number of other (neither 2xx, 3xx, 4xx, nor 5xx) responses issued by the server instance.
iwsInstanceCount200	Number of 200 (OK) responses issued by the server instance.
iwsInstanceCount302	Number of 302 (Moved Temporarily) responses issued by the server instance.
iwsInstanceCount304	Number of 304 (Not Modified) responses issued by the server instance.
iwsInstanceCount400	Number of 400 (Bad Request) responses issued by the server instance.
iwsInstanceCount401	Number of 401 (Unauthorized) responses issued by the server instance.
iwsInstanceCount403	Number of 403 (Forbidden) responses issued by the server instance.
iwsInstanceCount404	Number of 404 (Not Found) responses issued by the server instance.
iwsInstanceLoad1MinuteAverage	1 minute load average of the system running the server instance.
iwsInstanceLoad5MinuteAverage	5 minute load average of the system running the server instance.
iwsInstanceLoad15MinuteAverage	15 minute load average of the system running the server instance.
iwsInstanceNetworkInOctets	Number of octets transmitted on the network per second.
iwsInstanceNetworkOutOctets	Number of octets received on the network per second.
iwsVsTable	Server virtual servers.
iwsVsId	Virtual server identifier.
iwsVsRequests	Number of requests processed by the virtual server.
iwsVsInOctets	Number of octets received by the virtual server.
iwsVsOutOctets	Number of octets transmitted by the virtual server.

`appserv.mib` Managed Objects and Descriptions *(Continued)*

Managed object	**Description**
`iwsVsCount2xx`	Number of 200-level (Successful) responses issued by the virtual server.
`iwsVsCount3xx`	Number of 300-level (Redirection) responses issued by the virtual server.
`iwsVsCount4xx`	Number of 400-level (Client Error) responses issued by the virtual server.
`iwsVsCount5xx`	Number of 500-level (Server Error) responses issued by the virtual server.
`iwsVsCountOther`	Number of other (neither 2xx, 3xx, 4xx, nor 5xx) responses issued by the virtual server.
`iwsVsCount200`	Number of 200 (OK) responses issued by the virtual server.
`iwsVsCount302`	Number of 302 (Moved Temporarily) responses issued by the virtual server.
`iwsVsCount304`	Number of 304 (Not Modified) responses issued by the virtual server.
`iwsVsCount400`	Number of 400 (Bad Request) responses issued by the virtual server.
`iwsVsCount401`	Number of 401 (Unauthorized) responses issued by the virtual server.
`iwsVsCount403`	Number of 403 (Forbidden) responses issued by the virtual server.
`iwsVsCount404`	Number of 404 (Not Found) responses issued by the virtual server.
`iwsProcessTable`	Sun ONE Application Server processes.
`iwsProcessId`	Operating system process identifier.
`iwsProcessThreadCount`	Number of request processing threads.
`iwsProcessThreadIdle`	Number of request processing threads currently idle.
`iwsProcessConnectionQueueCount`	Number of connections currently in connection queue.
`iwsProcessConnectionQueuePeak`	Largest number of connections that have been queued simultaneously.
`iwsProcessConnectionQueueMax`	Maximum number of connections allowed in connection queue.
`iwsProcessConnectionQueueTotal`	Number of connections that have been accepted.

`appserv.mib` Managed Objects and Descriptions *(Continued)*

Managed object	Description
`iwsProcessConnectionQueueOverflows`	Number of connections rejected due to connection queue overflow.
`iwsProcessKeepaliveCount`	Number of connections currently in keepalive queue.
`iwsProcessKeepaliveMax`	Maximum number of connections allowed in keepalive queue.
`iwsProcessSizeVirtual`	Process size in kilobytes.
`iwsProcessSizeResident`	Process resident size in kilobytes.
`iwsProcessFractionSystemMemoryUsage`	Fraction of process memory in system memory.
`iwsListenTable`	Sun ONE Application Server listen sockets.
`iwsListenId`	Listen socket identifier.
`iwsListenAddress`	Address where socket listens.
`iwsListenPort`	Port where socket listens.
`iwsListenSecurity`	Encryption support.
`iwsThreadPoolCount`	Number of requests denied.
`iwsThreadPoolMax`	Maximum number of requests allowed in queue.
`iwsThreadPoolPeak`	Largest number of requests that have been queued simultaneously.
`iwsThreadPoolTable`	Sun ONE Application Server thread pools.
`iwsVsCount503`	Number of 503 (Unavailable) responses issued.
`iwsInstanceCount503`	Number of 503 (Unavailable) responses issued.

SNMP Messages

`GET` and `SET` are two types of messages defined by SNMP.

Each object is assigned a unique identifier within the MIB. Objects are accessed by the SNMP Manager by issuing `GET` and `GETNEXT` commands which specify the object's unique identifier. The Proxy Agent obtains the value of the specified object and transmits it to the SNMP manager. Events added to the log may generate SNMP traps provided they satisfy trap filter conditions. Events that do not generate traps are merely recorded as an entry in the maintenance log table and are accessed by the SNMP Manager through normal `GET` and `GETNEXT` commands.

GET and SET messages are sent by a network management station (NMS) to a master agent. You can use one or the other, or both, through the Administration interface.

SNMP exchanges network information in the form of protocol data units (PDUs). These units contain information about variables stored on the managed device, such as the HTTP server. These variables, also known as managed objects, have values and titles that are reported to the NMS as necessary. Protocol data units sent by the server to the NMS are known as *traps*. The use of GET, SET, and trap messages are explained further in the following sections.

SNMP Trap Destinations

An *SNMP trap* is a message the SNMP agent sends to a network management station (NMS). For example, an SNMP agent sends a trap when the status of an interface has changed from up to down. The SNMP agent must know the address of the NMS so it knows where to send traps.

You can configure this trap destination for the SNMP master agent from the Sun ONE Application Server Administration interface. You can also view, edit, and remove the trap destinations you have already configured. When you configure trap destinations using the Administration interface, you are actually editing the CONFIG file.

The server subagent sends a message or trap to the NMS when a significant event has occurred. For example:

1. The subagent informs the master agent that the server has stopped.
2. The master agent sends a message, or trap, reporting the event to the NMS.
3. The NMS displays the information textually or graphically through its network management application.

Refer to “Installing the SNMP Master Agent” on page 171 for instructions on setting up the SNMP trap port.

SNMP Agent Community

The *SNMP agent community* consists of a community string and the operations assigned to the specified community. The community string is a text string for a network management station (NMS) name that an SNMP agent uses for authorization. This means that an NMS would send a community string with each message it sends to the agent.

The operations assigned are get and/or set. The SNMP agent can then verify whether the NMS is authorized to perform get, set, or both get and set operations for data exchange. Community strings are not concealed when sent in SNMP packets; strings are sent in ASCII text.

You can configure and manage the community string and the allowed operations for each specified community from the Administration interface. Refer to "Installing the SNMP Master Agent" on page 171 for instructions on setting up the SNMP Agent Community.

Setting Up SNMP

In general, to use SNMP, a master agent and at least one subagent must be installed and running on your system. The master agent must be installed before you can enable a subagent. Refer to "Installing the SNMP Master Agent" on page 171.

The procedures for setting up SNMP are different depending upon your system. The following table provides an overview of procedures to follow for different situations. The actual procedures are described in detail later in the chapter.

If your server meets these conditions...	...follow these procedures. (discussed in detail in the following sections).
No native agent is currently running	1. Start the master agent. 2. Enable the subagent for each server installed on the system.
• Native agent is currently running • No SMUX • No need to continue using native agent	1. Stop the native agent when you install the master agent for your Administration Server. 2. Start the master agent. 3. Configure the SNMP subagent for each server instance.

If your server meets these conditions...	...follow these procedures. (discussed in detail in the following sections).
• Native agent is currently running	1. Install a proxy SNMP agent.
• No SMUX	2. Start the proxy SNMP agent.
• Needs to continue using native agent	3. Restart the native agent using a port number other than the master agent port number.
	4. Start the master agent.
	5. Enable the subagent for each server installed on the system.

Before you begin, you should verify two things:

- Is your system already running an SNMP agent (an agent native to your operating system)?
- If so, does your native SNMP agent support SMUX communication?

See your system documentation for information on how to verify this information.

NOTE After changing SNMP settings in the Admin Server, installing a new server, or deleting an existing server, you must perform the following steps:

- (Windows 2000) Restart the Windows SNMP service or reboot the machine.
- (UNIX) Restart the SNMP master agent and the SNMP subagent using the Admin Server.

The following topics are addressed in this section:

- Using a Proxy SNMP Agent (UNIX/Linux)
- Installing the SNMP Master Agent

Using a Proxy SNMP Agent (UNIX/Linux)

You need to use a proxy SNMP agent when you already have a native agent running, and you want to continue using it concurrently with a Sun ONE Application Server master agent. Before you start, be sure to stop the native master agent. (See your system documentation for detailed information.)

NOTE To use a proxy agent, you'll need to install it and then start it. You'll also have to restart the native SNMP master agent using a port other than the port the Sun ONE Application Server master agent is running on.

This section includes the following topics:

- Installing the Proxy SNMP Agent
- Starting the Proxy SNMP Agent
- Restarting the Native SNMP Daemon

Installing the Proxy SNMP Agent

If an SNMP agent is running on your system and you want to continue using the native SNMP daemon, follow the steps in these sections:

1. Install the SNMP master agent. See "Installing the SNMP Master Agent" on page 171.
2. Install and start the proxy SNMP agent and restart the native SNMP daemon. See "Using a Proxy SNMP Agent (UNIX/Linux)" on page 170.
3. Start the SNMP master agent. See "Enabling and Starting the SNMP Master Agent" on page 174.
4. Enable the subagent. See "Enabling the Subagent" on page 180.

To install the SNMP proxy agent, edit the `CONFIG` file (you can give this file a different name), located in *install_dir*`/lib/snmp/sagt` in the server root directory, so that it includes the listening port for the SNMP daemon. The proxy agent also needs to include the MIB trees and traps that the proxy SNMP agent will forward.

Here is an example of a `CONFIG` file:

```
    AGENT AT PORT 1161 WITH COMMUNITY public
    SUBTREES 1.3.6.1.2.1.1,
           1.3.6.1.2.1.2,
           1.3.6.1.2.1.3,
           1.3.6.1.2.1.4,
           1.3.6.1.2.1.5,
           1.3.6.1.2.1.6,
           1.3.6.1.2.1.7,
           1.3.6.1.2.1.8
    FORWARD ALL TRAPS;
```

Starting the Proxy SNMP Agent

To start the proxy SNMP agent, enter the following at the command prompt:

```
# sagt -c CONFIG&
```

Restarting the Native SNMP Daemon

After starting the proxy SNMP agent, restart the native SNMP daemon at the port you specified in the `CONFIG` file.

To restart the native SNMP daemon, enter the following at the command prompt:

```
# snmpd -P port_number
```

where *port_number* is the port number specified in the `CONFIG` file. For example, on the Solaris platform, using the port in the previously mentioned example of a `CONFIG` file, you would enter:

```
# snmpd -P 1161
```

Installing the SNMP Master Agent

NOTE You cannot use the Administration interface to install and start the master SNMP agent unless the server is running as `root`.

To install the master SNMP agent:

1. Log in as root.

2. Check whether an SNMP daemon (snmpd) is running on port 161.

   If no SNMP daemon is running, go to Step 4.

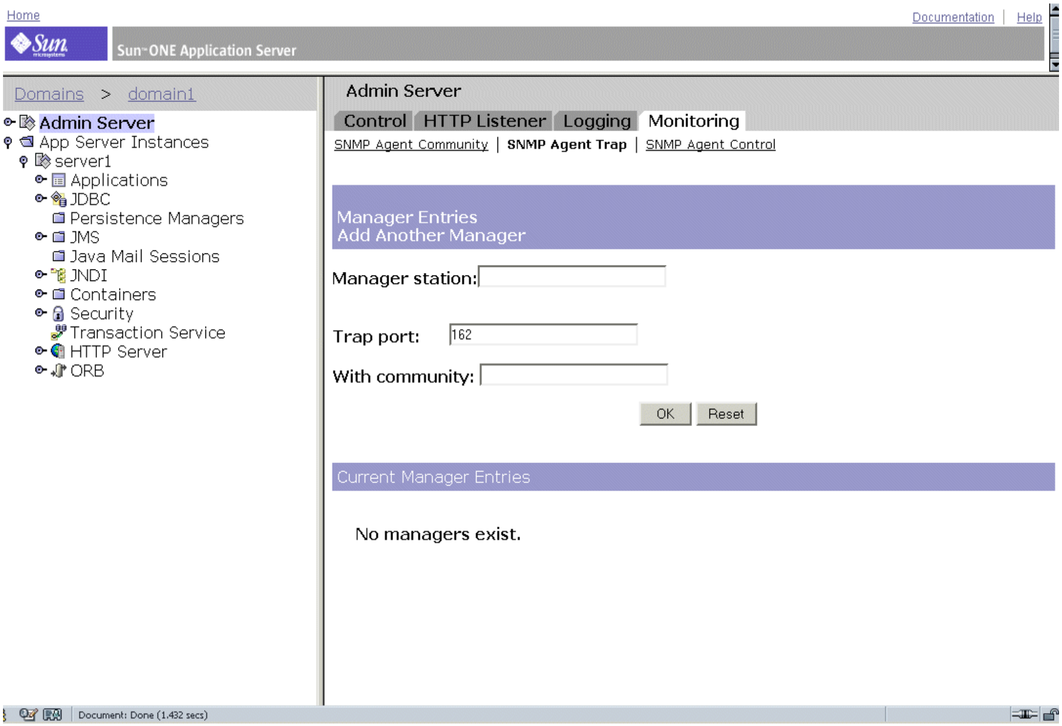
   If an SNMP daemon is running, make sure you know how to restart it and which MIB trees it supports.

3. If an SNMP daemon is running, kill its process.

4. In the Administration interface, select the Admin Server node from the left pane.

5. Select the Monitoring tab to display the SNMP Agent Trap page, as shown in the following figure:

Figure 6-2 SNMP Agent Trap Page



Admin Server

Manager Entries
Add Another Manager

Manager station:

Trap port: 162

With community:

Current Manager Entries

No managers exist.

The Manager Entries information is displayed on this page.

6. Type the name of the system that is running your network management software.

7. Type the trap port number at which your network management system listens for traps. (The well-known port is 162.) For more information on traps, see "SNMP Trap Destinations" on page 167.

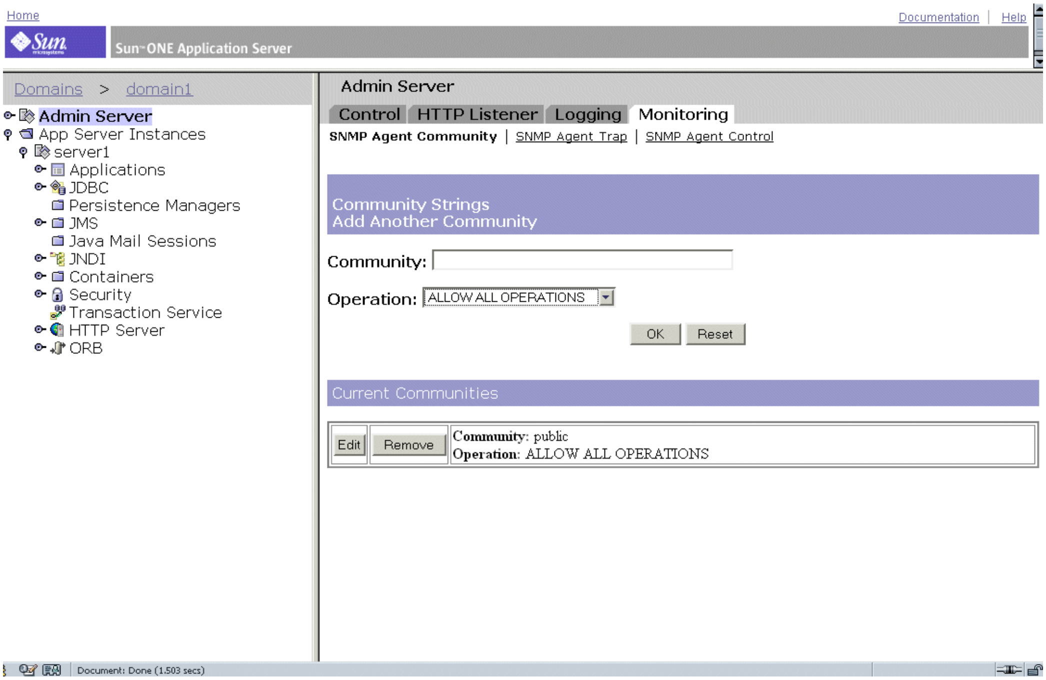
8. Type the community string you want to use in the trap. For more information on community strings, see "SNMP Agent Community" on page 167.

9. Click OK.

10. Click the SNMP Agent Community link within the Monitoring tab.

    The Community Strings information is displayed, as shown in the following figure.

Figure 6-3 SNMP Agent Community Page



Community: public
Operation: ALLOW ALL OPERATIONS

11. Type the community string for the master agent.

12. Choose an operation level for the community.

    After you establish a community, you can edit its settings or remove it from the buttons indicated within the Current Communities heading on this page.

13. Click OK.

14. Access App Server Instances and your server instance in the left pane, then click Apply Changes.

Enabling and Starting the SNMP Master Agent

Master agent operation is defined in an agent configuration file named `CONFIG`, which you can edit manually. You must install the master SNMP agent before you can enable the SNMP subagent.

NOTE If you get a bind error similar to `System Error: Could not bind to port` when restarting the master agent, use `ps -ef | grep snmp` to check if `magt` is running. If it is running, use the command `kill -9` *pid* to end the process. The CGIs for SNMP will then start working again.

This section includes the following topics:

- Starting the Master Agent on Another Port
- Manually Configuring the SNMP Master Agent
- Editing the Master Agent CONFIG File
- Defining sysContact and sysLocation Variables
- Configuring the SNMP Subagent
- Starting the SNMP Master Agent
- Enabling the Subagent

Starting the Master Agent on Another Port

The Administration interface will not start the SNMP master agent on ports other than 161. However, you can manually start the master agent on another port using the following steps:

1. Edit *install_dir*`/lib/snmp/magt/CONFIG` to specify the desired port.

2. Run the start script as follows:

   ```
   cd instance_root/admin-server ./start -shell
   install_dir/lib/snmp/magt/magt
   install_dir/lib/snmp/magt/CONFIG
   install_dir/lib/snmp/magt/INIT
   ```

The master agent will then start on the desired port. However, the Administration interface will be able to detect that the master agent is running.

Manually Configuring the SNMP Master Agent

To configure the master SNMP agent manually:

1. Log in as root.
2. Check to see if there is an SNMP daemon (`snmpd`) running on port 161.

   If an SNMP daemon is running, make sure you know how to restart it and which MIB trees it supports. Then kill its process.
3. Edit the `CONFIG` file located in `lib/snmp/magt` in the server root directory.
4. (Optional) Define `sysContact` and `sysLocation` variables in the `CONFIG` file as described in "Defining sysContact and sysLocation Variables" on page 176.

Editing the Master Agent CONFIG File

The `CONFIG` file defines the community and the manager that will work with the master agent. The manager value should be a valid system name or an IP address.

Here is an example of a basic `CONFIG` file:

```
COMMUNITY           public
                    ALLOW ALL OPERATIONS

MANAGER             manager_station_name
                    SEND ALL TRAPS TO PORT 162
                     WITH COMMUNITY public
```

Defining sysContact and sysLocation Variables

You can edit the `CONFIG` file to add initial values for `sysContact` and `sysLocation` which specify the `sysContact` and `sysLocation` MIB-II variables. The strings for `sysContact` and `sysLocation` in this example are enclosed in quotes. Any string that contains spaces, line breaks, tabs, and so on must be in quotes. You can also specify the value in hexadecimal notation.

Here is an example of a `CONFIG` file with `sysContact` and `sysLocation` variables defined:

```
COMMUNITY          public
                   ALLOW ALL OPERATIONS

MANAGER            nms2
                   SEND ALL TRAPS TO PORT 162
                   WITH COMMUNITY public

INITIAL            sysLocation “Server room
901 San Antonio Road
Palo Alto CA 94303
USA”

INITIAL            sysContact “John Doe
email: jdoe@sun.com”
```

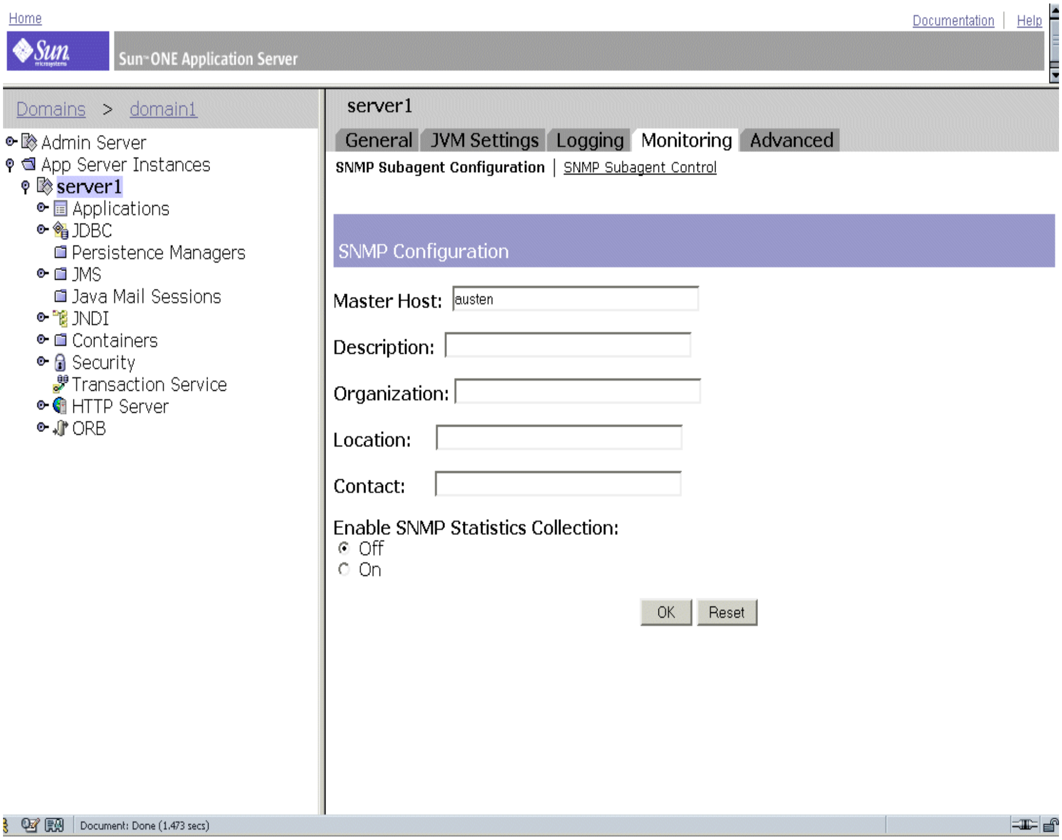
Configuring the SNMP Subagent

To configure the SNMP subagent, perform the following steps:

1. From the Admin Server, select the server instance node in the left pane.
2. Select the Monitoring tab from the right pane.
3. Select the SNMP Subagent Configuration link.

   The following page is displayed:

Figure 6-4 SNMP Subagent Configuration Page



SNMP Configuration

Master Host: austen

Description:

Organization:

Location:

Contact:

Enable SNMP Statistics Collection:
● Off
○ On

4. (UNIX only) Enter the name and domain of the server in the Master Host field.

5. Enter the Description of the server, including operating system information.

6. Enter the Organization responsible for the server.

7. Enter the location of the server instance.

8. Enter the name of the person responsible for the server and the person's contact information in the Contact field.

9. Select On to Enable the SNMP Statistics Collection.

10. Click OK.

11. Access App Server Instances and your server instance in the left pane, then click Apply Changes.

Starting the SNMP Master Agent

Once you have installed the SNMP master agent, you can start it manually or by using the Admin Server from the Administration interface.

Manually Starting the SNMP Master Agent

To start the master agent manually, enter the following at the command prompt:

```
# magt CONFIG INIT&
```

The `INIT` file is a nonvolatile file that contains information from the MIB-II system group, including system location and contact information. If `INIT` doesn't already exist, starting the master agent for the first time will create it.

NOTE An invalid manager name in the `CONFIG` file will cause the master agent start-up to fail.

To start a master agent on a non-standard port, use one of two methods:

Method 1: In the `CONFIG` file, specify a transport mapping for each interface over which the master agent listens for SNMP requests from managers. Transport mappings allow the master agent to accept connections at the standard port and at a non-standard port. The master agent can also accept SNMP traffic at a non-standard port. The maximum number of concurrent SNMP is limited by your target system's limits on the number of open sockets or file descriptors per process. Here is an example of a transport mapping entry:

```
TRANSPORT          extraordinary   SNMP
                   OVER UDP SOCKET
                    AT PORT 11161
```

After editing the `CONFIG` file manually, you should start the master agent manually by typing the following at the command prompt:

```
# magt CONFIG INIT&
```

Method 2: Edit the `/etc/services` file to allow the master agent to accept connections at the standard port as well as at a non-standard port.

Starting the SNMP Master Agent Using the Admin Server

To start the SNMP master agent using the Admin Server, perform the following steps:

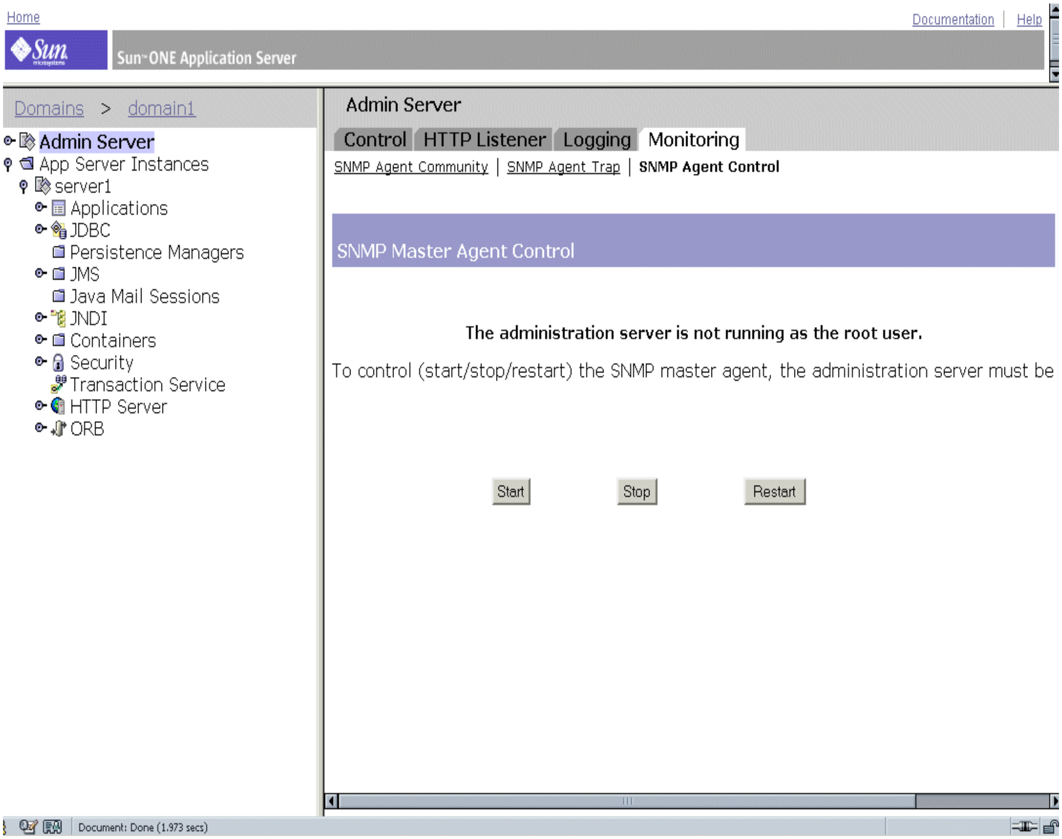
NOTE You must be logged in the Sun ONE Application Server as root to start the SNMP Master Agent.

---

1. Log in to the Admin Server.
2. From the Admin Server node in the left pane, choose the Monitoring tab.
3. Choose the SNMP Agent Control link near the top of the right pane.

   The following page is displayed.

Figure 6-5 SNMP Agent Control Page



The administration server is not running as the root user.

To control (start/stop/restart) the SNMP master agent, the administration server must be

4. Click Start.

   You can also stop and restart the SNMP master agent from the SNMP Agent Control page.

Enabling the Subagent

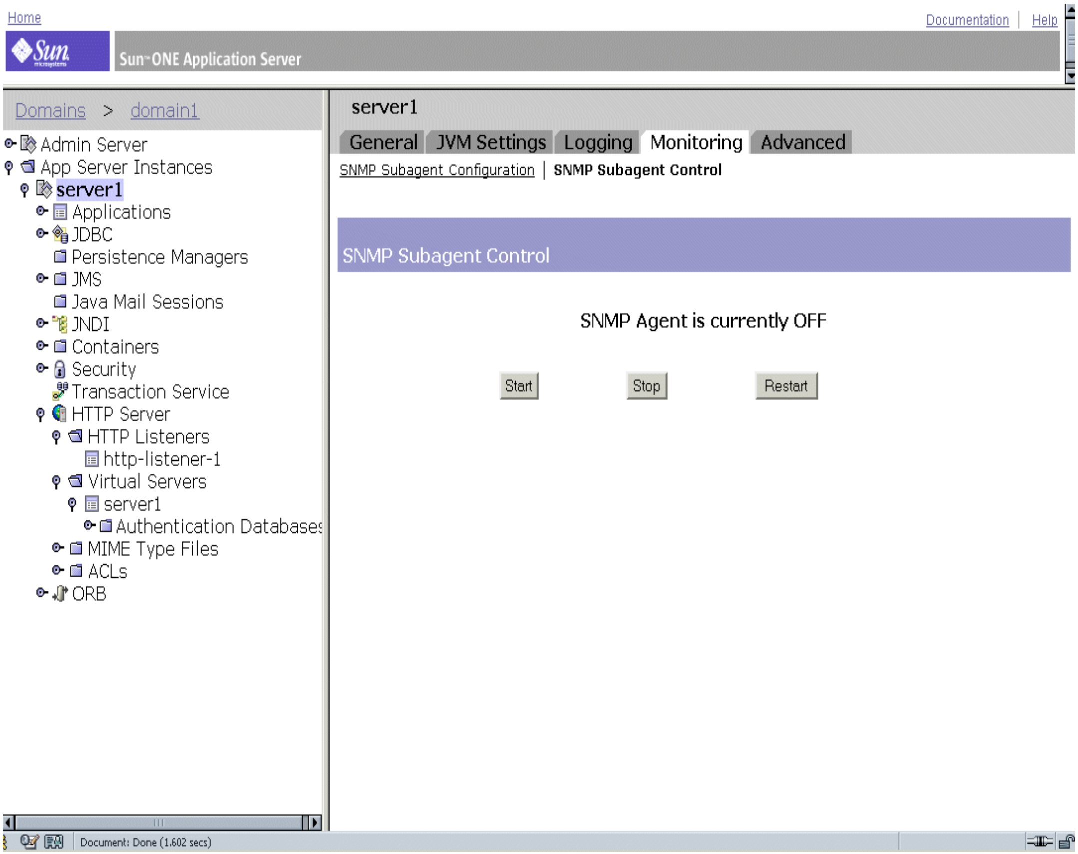
After you have installed the master agent that comes with the Administration Server, you must enable the subagent for your server instance before you attempt to start it. For information on installing the master agent, see "Installing the SNMP Master Agent" on page 171.

You can use the subagent to stop the SNMP function on UNIX/Linux platforms. You must stop the subagent first, then the master agent. If you stop the master agent first, you may not be able to stop the subagent. If that happens, restart the master agent, stop the subagent, then stop the master agent.

To enable the SNMP subagent:

1. Expand the App Server Instances node in the left pane.
2. Select the server instance, then click the Monitoring tab.
3. Select the SNMP Subagent Control option to display the page as shown in the following figure.

Figure 6-6 SNMP Subagent Control Page



SNMP Agent is currently OFF

From this page, you can start, stop, or restart the SNMP subagent. The status of the subagent is indicated just above the control buttons.

On the Windows platform, the Windows SNMP service is used for monitoring the Sun ONE Application Server; it can be controlled from the Control Panel/Administrative Tools/Services option.

NOTE After making any SNMP configuration changes, you must click OK, then restart SNMP subagent from the SNMP Subagent Control page.

Chapter 7

Configuring the Web Server Plugin

This chapter explains how Sun ONE Application Server processes HyperText Transfer Protocol (HTTP) requests, and how to configure and use the web server plugin with Sun ONE Application Server. This chapter also explains how to configure and use the web server plugin with Microsoft IIS, and the Apache web server.

This chapter includes the following topics:

- About the Web Server Plugin
- Handling Client Requests
- Web Server Plugin Configuration
- The Web Server Plugin SAF Reference
- Using the Web Server Plugin
- Configuring Microsoft IIS To Use the Web Server Plugin
- Configuring Apache Web Server

About the Web Server Plugin

The web server plugin is an HTTP reverse proxy plugin that allows you to instruct a Sun ONE Web Server or Sun ONE Application Server to forward certain HTTP requests to another server. For example, you can configure a web server connected to the Internet to forward requests for specific web applications to an application server located behind a corporate firewall.

Within Sun ONE Application Server, the web server plugin allows one server instance to forward an HTTP (web) request to another server instance.

The web server plugin performs the following functions:

- Re-uses connections from the proxy server, whenever possible. This eradicates the necessity of opening new connections to process incoming requests.
- The web server plugin starts streaming requests and responses as it starts receiving them. In other words, the plugin does not wait till the request or response is collected fully before forwarding it to the remote server.
- The web server plugin maintains multiple outbound HTTP connections to the same remote server, as appropriate. Connections formed for requests that are forwarded by the web server plugin are called outbound HTTP connections.

To understand how the web server plugin works, it is necessary to understand the basics of HTTP requests, and specifically the method used by Sun ONE Application Server to process HTTP requests.

Handling Client Requests

Sun ONE Application Server is an application server that can directly accept and respond to HTTP requests. In this section, we will discuss HTTP basics in and also look at how Sun ONE Application Server handles requests. This section covers the following topics:

- HTTP Basics
- Steps in the Request Handling Process

HTTP Basics

As a quick summary, the HTTP/1.1 protocol works as follows:

- The client (usually a browser) opens a connection to the server and sends a request
- The server processes the request, generates a response, and closes the connection if it finds a `Connection: Close` header.

The request consists of a line indicating a method such as `GET` or `POST`, a Universal Resource Identifier (URI) indicating which resource is being requested, and an HTTP protocol version separated by spaces.

This is normally followed by a number of headers, a blank line indicating the end of the headers, and sometimes body data. Headers may provide various information about the request or the client body data. Headers are typically only sent for `POST` and `PUT` methods.

The example request shown below would be sent by a browser to request the server `foo.com` to send back the resource in */index.html*. In this example, no body data is sent because the method is `GET` (the point of the request is to get some data, not to send it.)

```
GET /index.html HTTP/1.0

User-agent: Mozilla

Accept: text/html, text/plain, image/jpeg, image/gif, */*

Host: foo.com
```

The server receives the request and processes it. It handles each request individually, although it may process many requests simultaneously. Each request is broken down into a series of steps that together make up the request handling process.

The server generates a response which includes the HTTP protocol version, HTTP status code, and a reason phrase separated by spaces. This is normally followed by a number of headers. The end of the headers is indicated by a blank line. The body data of the response follows. A typical HTTP response might look like this:

```
HTTP/1.0 200 OK
Server: Standard/7.0
Content-type: text/html
Content-length: 83

<HTML>
<HEAD><TITLE>Hello World</Title></HEAD>
<BODY>Hello World</BODY>
</HTML>
```

The status code and reason phrase tell the client how the server handled the request. Normally the status code 200 is returned indicating that the request was handled successfully and the body data contains the requested item. Other result codes indicate redirection to another server or the browser’s cache, or various types of HTTP errors such as “404 Not Found.”

Steps in the Request Handling Process

When Sun ONE Application Server first starts up it performs certain initialization tasks, and then waits for an HTTP request from a client (such as a browser). When it receives a request, it first selects a virtual server.

After the virtual server is selected, the virtual server's `obj.conf` file specifies how the request is handled with the following steps:

1. **AuthTrans** (authorization translation)

   Verify any authorization information (such as name and password) sent in the request.

2. **NameTrans** (name translation)

   Translate the logical URI into a local file system path.

3. **PathCheck** (path checking)

   Check the local file system path for validity and check that the requestor has access privileges to the requested resource on the file system.

4. **ObjectType** (object typing)

   Determine the MIME-type (Multi-purpose Internet Mail Encoding) of the requested resource (for example. `text/html`, `image/gif`, and so on).

5. **Service** (generate the response)

   Generate and return the response to the client.

6. **AddLog** (adding log entries)

   Add entries to log file(s).

7. **Error** (service)

   This step is executed only if an error occurs in the previous steps. If an error occurs, the server logs an error message and aborts the process.

Web Server Plugin Configuration

The configuration and behavior of the web server plugin are determined by a set of configuration files. Sun ONE Application Server looks at the configuration defined in these files each time it processes a request from a client. The configuration files are named `obj.conf` and `init.conf`. The `obj.conf` file is prefixed with the name of the virtual server, for example `server1-obj.conf`. For more information, see "The obj.conf File" on page 373.

Each instance of Sun ONE Application Server has its own `init.conf` file, to which the server refers at startup.

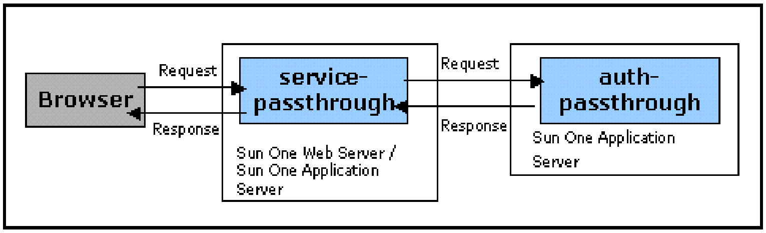
As discussed in the preceding topic, the `obj.conf` configuration file contains a series of instructions (directives) that tell Sun ONE Application Server what to do at each stage in the client request and response process. Each directive invokes a Server Application Function (SAF).

The `obj.conf` file is essential to the operation of the Sun ONE Application Server. When you make changes to the server through the Administration interface, the system automatically updates `obj.conf`.

The `init.conf` configuration file sets values of variables that configure the server during initialization. The server executes the configuration parameters specified in this file, during server start up. See the *Sun ONE Application Server Administrator's Configuration File Reference* for more information.

The following figure illustrates the relationship between the web browser, a front-end web server, a backend application server, and the web server plugin's `service-passthrough` and `auth-passthrough` SAFs:

Figure 7-1 Relationship Between Web Browser, Web Server, Application Server, and Web Server Plugin SAFs


Browser
Request
Response
service-passthrough
Sun One Web Server / Sun One Application Server
Request
Response
auth-passthrough
Sun One Application Server

The Web Server Plugin SAF Reference

This section discusses the function and behavior of the following Server Application Functions (SAF):

- init-passthrough
- auth-passthrough
- service-passthrough
- check-passthrough

init-passthrough

The `init-passthrough` function initializes the web server plugin. This function must be called before the web server plugin can be used.

Example:

```
Init fn="load-modules" shlib="c:/plugins/passthrough.dll"
funcs="init-passthrough,auth-passthrough,check-passthrough,service-
passthrough" NativeThread="no"
```

```
Init fn="init-passthrough"
```

auth-passthrough

The `auth-passthrough` SAF is applicable in `AuthTrans-class` directives.

The auth-passthrough function inspects the incoming HTTP (web) request for client information encoded by a service-passthrough function running on an intermediate server. The client information includes:

- IP address request originated from
- SSL key size used by originating client
- SSL client certificate presented by originating client

When auth-passthrough detects encoded client information, it treats the request as a direct request from the originating client and not as a request forwarded by an intermediate server running `service-passthrough`.

This is useful in two-tier deployment scenarios where;

- a Sun ONE Application Server instance is hidden by a second firewall behind the corporate firewall
- no client connections are permitted directly to the S1AS instance.

In such network architectures, a client always connects to the front-end web server which is running the proxy plugin. This web server is the one that forwards requests to Sun ONE Application Server. This indicates that Sun ONE Application Server can only receive requests from the proxy host (in this case, the web server), and never directly from client hosts. This means that if applications, deployed on a Sun ONE Application Server instance that is behind two firewalls, query for client information such as the client's IP address, the applications will get the proxy host IP (since that is the actual originating host for the relayed request). The `auth-passthrough` SAF can be used to modify this behavior so that the remote (proxied) client information is presented instead.

Since `auth-passthrough` makes it possible to override information that may be used for authentication (for example, the IP address from which the request originated) it is important that only trusted clients or servers be allowed to connect to a server running `auth-passthrough`. As a precautionary measure, it is recommended that only servers behind the corporate firewall should run `auth-passthrough`. A server that is accessible through the Internet should not run the `auth-passthrough` SAF. The `auth-passthrough` SAF should be used only if the relevant information about the originating client is required.

Note that in the scenario described above, SSL client authentication could be turned 'on' only for web server and turned 'off' always for application server, for the configuration to work properly.

Command Example:

```
AuthTrans fn="auth-passthrough"
```

service-passthrough

The `service-passthrough` `SAF` is applicable in Service-class directives.

The `service-passthrough` SAF forwards a request from one server to another server for processing. The `service-passthrough` SAF can be configured to use SSL or non SSL (HTTPS or HTTP) connections to the remote server, independent of the type of connection through which the original request was received. The `service-passthrough` SAF encodes information about the originating client that may be decoded by an `auth-passthrough` function running on the remote server.

A `service-passthrough` directive is typically used in combination with other directives in the `obj.conf` configuration file as follows:

```
<Object name="passthrough">

ObjectType fn="force-type" type="magnus-internal/passthrough"

Error reason="Bad Gateway" fn="send-error"
uri="$docroot/badgateway.html"

</Object>

<Object name="default">

....

NameTrans fn="assign-name" from="(/webapp1|/webapp1/*)" name="passthrough"

...

</Object>
```

If the backend application server is down, the user will be shown the local HTML file `badgateway.html` instead. In case the server running the `service-passthrough` SAF needs to serve files to which it has access, and forward only rejected requests to the backend application servers, the `ObjectType` line would be changed to:

```
ObjectType fn="check-passthrough" type="magnus-internal/passthrough"
```

check-passthrough

The `check-passthrough` SAF is Applicable in `ObjectType-class` directives.

The check-passthrough function checks to see if the requested resource (for example, an HTML document, or a GIF image) is available on the local server. If the requested resource does not exist locally, the `check-passthrough` SAF sets the type to indicate that the request should be passed to another server to be processed by the `service-passthrough` SAF.

Parameters:

`type` - (optional) type to set when the request resource does not exist. The default is "`magnus-internal/passthrough`".

Example

```
ObjectType fn="check-passthrough"
```

Using the Web Server Plugin

To use the web server plugin on a Sun ONE Web Server, you must make changes to the configuration files of both Sun ONE Application Server and Sun ONE Web Server. Follow the procedures listed in this section to configure and use the Sun ONE Web Server plugin:

- Changes to Sun ONE Web Server
- Changes to Sun ONE Application Server

Changes to Sun ONE Web Server

Take backups of critical configuration files, such as `magnus.conf` and `obj.conf`, before making changes to these files.

1. Create a directory in the web server installation area that will contain the passthrough plugin. For example:

   `cd /`*webserver_install_dir*`/plugins`

   `mkdir -p passthrough/bin`

2. Copy the passthrough plugin from Sun ONE Application Server installation to this new, web server directory. For example:

   `cd` *appserver_install_dir*`/lib`

   `cp libpassthrough.so` *webserver_install_dir*`/plugins/passthrough/bin`

   For Windows, copy the `passthrough.dll` file.

3. Edit the `magnus.conf` file, found under *webserver_install_dir/https-host.domain*`/config`, and append the following lines:

```
Init fn="load-modules"
shlib="webserver_install_dir/plugins/passthrough/bin/libpassthrough.so"
funcs="init-passthrough,auth-passthrough,check-passthrough,service-passthrough"
NativeThread="no"

Init fn="NSServletEarlyInit" EarlyInit=yes

Init fn="NSServletLateInit"  LateInit=yes

Init fn="init-passthrough"
```

4. Edit the `obj.conf` file, found under *webserver_install_dir/https-host.domain*`/config`, and add the `NameTrans` directive as shown:

```
<Object name=default>
NameTrans fn="NSServletNameTrans" name="servlet"
NameTrans fn="assign-name" from="/*" name="passthrough"
</Object>
```

The `from="/*" uri` is the context root of a web application deployed on the remote servers, and `passthrough` corresponds to the name of the `<Object>` in `obj.conf`.

For example:

```
<Object name="default">
...
NameTrans fn="assign-name" from="(/webapp1|/webapp1/*)"
name="passthrough"
...
</Object>
```

5. Add the following lines in the `obj.conf` file:

```
<Object name="passthrough">
ObjectType fn="force-type" type="magnus-internal/passthrough"
PathCheck fn="deny-existence" path="*/WEB-INF/*"
Service type="magnus-internal/passthrough" fn="service-passthrough"
servers="http://servername:port"
Error reason="Bad Gateway" fn="send-error" uri="$docroot/badgateway.html"
</Object>
```

6. Restart the Sun ONE Web Server instance.

Next, you must configure Sun ONE Web Server to route requests to the web server plugin.

Changes to Sun ONE Application Server

Take backups of the critical configuration files, such as `magnus.conf` and `obj.conf`, before making changes to these files.

1. Edit the *install_dir*`/domains/domain1/server1/config/init.conf` file and append the following lines:

   On UNIX:

```
Init fn="load-modules"
shlib="webserver_install_dir/plugins/passthrough/bin/libpassthrough.so"
funcs="init-passthrough,auth-passthrough,check-passthrough,service-passthrough"
NativeThread="no"
Init fn="init-passthrough"
```

On Windows:

```
Init fn="load-modules" shlib="c:/install_dir/bin/passthrough.dll"
funcs="init-passthrough,auth-passthrough,check-passthrough, service-passthrough"
NativeThread="no"
Init fn="init-passthrough"
```

2. Edit the *install_dir*`/domains/domain1/server1/config/server1-obj.conf` and add the `AuthTrans` directive as shown here:

```
<Object name="default">
AuthTrans fn="match-browser" browser="*MSIE*" ssl-unclean-shutdown="true"
AuthTrans fn="auth-passthrough"
fn="service-passthrough" servers="server"
Error reason="Bad Gateway" fn="send-error" uri="$docroot/badgateway.html"
</Object>
```

server is a URL of the following form:

`http://`*servername*`:`*port*

3. Restart the Sun ONE Application Server instance.

This completes the changes you need to make to Sun ONE Application Server. The procedures in the preceding section is to configure a single instance of the application server. To enable more instances to use the web server plugin, you must make these changes in the configuration files of those application server instances.

NOTE For Solaris and Linux, the name of the plugin library will be `libpassthrough.so`. For Windows, the name of the plugin library will be `passthrough.dll`

Configuring Microsoft IIS To Use the Web Server Plugin

Configuring the Microsoft Internet Information Services to use the web server plugin involves configuring the web server plugin for use with Microsoft IIS, and configuring Microsoft IIS to use the web server plugin.

You can also configure server pools to handle multiple applications that run on different servers.

The following topics are covered in this section:

- Configuring the Web Server Plugin for IIS
- Configuring IIS to Use the Web Server Plugin
- Configuring Multiple Server Pools
- Sample sun-passthrough.properties File

Configuring the Web Server Plugin for IIS

To configure the web server plugin for IIS, perform the following tasks:

1. Create a directory for the web server plugin under the under the IIS `wwwroot` directory, by typing the following command, from the C:\ command line prompt:

   ```
   md \Inetpub\wwwroot\sun-passthrough
   ```

2. Copy the plugin files to the `C:\Inetpub\wwwroot\sun-passthrough` directory.

3. Use a text editor to add the URL of the machine on which Sun ONE Application Server is installed, to the `C:\Inetpub\wwwroot\sun-passthrough\sun-passthrough.properties` file.

   You will need to add the following information via a text editor:

   `server=http://`*appservername*`:`*port*

   where, *appservername* is the hostname or IP address of machine on which Sun ONE Application Server is installed, and `port` is the number of the port on which it listens (this value is typically set to 80).

4. List the context roots you want Sun ONE Application Server to service in the `C:\Inetpub\wwwroot\sun-passthrough\sun-passthrough.properties` file.

   These context roots should correspond to the context roots of applications deployed on Sun ONE Application Server. Requests to these context roots will be serviced by Sun ONE Application Server, while other requests are handled by the IIS web server. The command lines to pass requests to a web application is:

   `passthrough=/`*webapplication*

   where, `/`*webapplication* is the context root of a web application. To pass all requests to Sun ONE Application Server, add the following line:

   `passthrough=/`

You have now configured the web server plugin in the Microsoft IIS root directory. To complete the process, you now need to configure Microsoft IIS to use the web server plugin.

Configuring IIS to Use the Web Server Plugin

To configure IIS to use the web server plugin, you need to open the Windows Internet Services Manager. The Internet Services Manager is located in the Administrative Tools folder in the Control Panel folder.

Open the Internet Services Manager, and perform the following tasks:

1. Select the web site for which you want to enable the plugin. This web site is typically named the Default Web Site.
2. Right click on the web site and select Properties to open the Properties notebook.
3. Open the ISAPI Filters tab, click the Add button, and follow the steps given below, to add a new ISAPI filter:
   a. In the Filter Name field, enter Sun ONE Application Server
   b. In the Executable field, type C:\Inetpub\wwwroot\sun-passthrough\sun-passthrough.dll
   c. Click OK, and close the Properties notebook.
4. You now need to create and configure a new virtual directory. Follow the steps given below to create and configure a new virtual directory:

a. Right click on the default web site, select New, and then Virtual Directory. The Virtual Directory Creation Wizard opens.

b. In the Alias field, type `sun-passthrough`.

c. In the Directory field, type `C:\Inetpub\wwwroot\sun-passthrough`

d. Ensure that you check the Execute Permission checkbox and that all other permission checkboxes are left unchecked.

e. Click Finish.

5. You need to stop and start the web server, for your new settings to take effect. To stop the web server, right click on the web site and select Stop. To start the web server, right click on the web site and select Start.

   Next, type the following in a web browser, to access the web application context root:

   `http://`*webservername*`/`*webapplication*

   where, *webservername* is the hostname or IP address of the web server and `/`*webapplication* is the context root that you listed in the `C:\Inetpub\wwwroot\sun-passthrough\sun-passthrough.properties` file to verify that the web server, web server plugin, and Sun ONE Application Server are operating correctly.

Configuring Multiple Server Pools

It is possible to partition your web applications across multiple application servers (that is, you run some applications on one set of servers and other applications on another set of servers) by configuring server pools in the `sun-passthrough.properties` file. For each server pool, choose a unique name, comprised of letters and numbers. Once you complete the steps for installing and configuring the web server plugin for Microsoft IIS, as given in the section “Configuring Microsoft IIS To Use the Web Server Plugin” on page 194, edit the `C:\Inetpub\wwwroot\sun-passthrough\sun-passthrough.properties` file and prefix the relevant server and passthrough property lines with the unique name that you choose for the server pool. Place a period (.) after the server pool name.

For example, the following lines from the `sun-passthrough.properties` file define two server pools. The first server pool consists of `server-a` and the services requests for context root `/app1`. The second server pool consists of `server-b` and the services requests for `/app2` and `/app3` context roots.

```
server=http://server-a
```

```
passthrough=/app1
serverpool2.server=http://server-b
serverpool2.passthrough=/app2
serverpool2.passthrough=/app
```

Sample sun-passthrough.properties File

```
# Sun ONE Application Server web server plugin for IIS
#
# This file is used to configure the Sun ONE Application Server web server
# plugin for IIS.  Lines beginning with a '#' are ignored.
# server
#
# The server property specifies the URL of an application server.  If multiple
# server properties are given, the plugin will distribute load across the
# specified application servers.
#
server=http://localhost:8080
# passthrough
#
# The passthrough property specifies the context root (virtual directory) of a
# web application.  Requests for the given context root will be passed to the
# application server for processing.  If 'passthrough=/' is specified, all
# requests will be passed to the application server for processing.
#
# passthrough properties should be ordered from most to least specific.  For
# example, 'passthrough=/apps/app1' should appear before 'passthrough=/apps'.
#
# Multiple passthrough properties are allowed.
#
#passthrough=/webapp
```

```
#passthrough=/servlets
#passthrough=*.jsp
passthrough=/
# prefix
#
# The prefix property specifies the IIS virtual directory that contains the
# plugin DLL, sun-passthrough.dll.
#
prefix=/sun-passthrough
# error-url
#
# The error-url property specifies the URL of a page to redirect the client to
# when the application server is unavailable.
#
#error-url=/badgateway.htm
# It is possible to configure multiple server pools by prefixing the server
# and passthrough property names with a pool name followed by a period ('.').
# Pool names can be any sequence of letters and numbers.
#
# For example, the following properties define two server pools.  One server
# pool will service the web applications at '/app1' and the other will service
# the web applications at '/app2' and '/app3':
#
#serverpool1.server=http://server-a
#serverpool1.passthrough=/app1
#
#serverpool2.server=http://server-b
#serverpool2.passthrough=/app2
#serverpool2.passthrough=/app3
```

Configuring Apache Web Server

This section describes how you can compile the Apache source code and configure your installation of Apache Web Server to work with Sun ONE Application Server.

To enable HTTP requests to be sent to Sun ONE Application Server, you must compile the Apache sources to use the `mod_proxy` module, and then modify the `httpd.conf` file.

This section contains information on the following procedures:

- Minimum Requirements
- Compiling Apache With mod_proxy Module
- Modifying the httpd.conf File
- Starting And Stopping Apache

Minimum Requirements

You must meet the following requirements to successfully compile Apache web server and use the `mod_proxy` plugin.

- `apache_1.3.27` (sources)
- Sun's C Compiler for Solaris 8 and 9.
- Sun ONE Application Server 7 installables.
- On Solaris 8 and 9, ensure that `CC` and `make` are in the `PATH`.

Compiling Apache With mod_proxy Module

1. Download the latest Apache source distribution with built-in mod_proxy module from `www.apache.org`

   Unpack the source distribution. The source distribution comes as a compressed archive. If you are installing Apache 1.3.27, the source distribution archive will read `apache_1.3.27.tar.gz`.

2. De-compress and untar the archive using the following command:

   ```
   $ tar -zxvf apache_1.3.27.tar.gz
   ```

   This command will create a directory called `apache_1.3.27` in your current working directory.

3. You now need to configure your environment to compile the Apache source code. The source distribution comes with a script called `configure`, which checks your environment for the necessary support files (such as headers, shared libraries and utility programs) that are required to successfully compile Apache.

   To configure you environment, go to the Apache source directory and proceed with the following steps:

   a. Make sure that the following paths exist, while installing Apache on Solaris:

      - `CC=/opt/SUNWspro/bin/cc`

        Where `/opt/SUNWspro/bin` is the path where `cc` is installed. Make sure that this is in your `PATH`.

      - Make sure that `/usr/ccs/bin` is in your `PATH`.

   b. Run the following command:

```
./configure --enable-module=proxy --prefix=/usr/local/apache
```

   The path specified in the `prefix` argument indicates where you wish to install Apache.This is a variable and you can specify the path where you want to install Apache.

   This command will output several lines on the screen.Essentially this command creates the Makefiles for the build according to your system configuration. If there are errors in `configure`, you may be missing some header files or utility programs which you must install before proceeding.

4. After the `configure` script runs successfully, you can compile Apache using the `make` command, as follows:

```
make
```

   This command will output several lines on the screen indicating that the process is compiling Apache source code and linking Apache. This process should normally conclude without errors. However if any errors occur, please check if all the library files and utility programs of Apache have been properly downloaded.

5. You now need to install Apache. Apache installs itself in the /usr/local/apache directory (or any other directory that you specify). To install Apache, run the following command:

   ```
   make install
   ```

   If this command executes successfully, your system now has Apache installed. You should see Apache's installation files in the following directory:

   ```
   /usr/local/apache
   ```

   The main configuration file, called httpd.conf, will be installed in the /usr/local/apache/conf directory.

NOTE mod_proxy lacks the SSL transport and load balancing features of Sun ONE Application Server 7 reverse proxy plugin. Also, it is not possible for web applications to obtain the client's IP address or SSL client certificate when using mod_proxy.

Modifying the httpd.conf File

Apache is configured through the file, httpd.conf. This file consists of a number of Apache directives, which determine the various operating parameters of the Apache server. For a simple installation of Apache, you will need to modify the following few directives:

```
ServerRoot “/usr/local/apache”
Port 5000
```

ServerRoot is the path in which you installed Apache.

Apache is now configured for default behavior and web serving. Next, you must add the following application server specific directives to the httpd.conf file to enable Apache to forward HTTP requests to Sun ONE Application Server:

```
<IfModule mod_proxy.c>
ProxyPass / http://<s1as_server.some.domain>:<port>/
ProxyPassReverse  / http://<s1as_server.some.domain>:<port>/
</IfModule>
```

Here, *<s1as_server.some.domain>*:*<port>* should be replaced by the URL address of your Sun ONE Application Server. Duplicate these two lines for each web application context root, where */application* is the web application context root and http://*server* is the URL of the Sun ONE Application Server.

This completes your configuration of Apache web server.

Starting And Stopping Apache

Apache comes bundled with a script titled `apachectl` that facilitates starting, stopping and restarting Apache. Run the follow command to start Apache

```
$ /usr/local/apache/bin/apachectl start
```

To stop apache, run the following command:

```
use /usr/local/apache/bin/apachectl stop
```

After you start, you can test your installation of Apache. Once Apache is running, type the following address in your web browser: `http://localhost/`. If your installation was successful and Apache is running, you should see a test page displaying a message to the effect.

Chapter 8

Configuring J2EE Containers

Sun ONE Application Server provides various J2EE containers in compliance with the J2EE 1.3 specification. Containers provide runtime support for J2EE application components such as Enterprise Java Beans (EJBs) and Message Driven Beans (MDBs). MDBs and EJBs never interact directly with other J2EE application components. They use the protocols and methods of the EJB container for interacting with each other and with platform services, such as the Java Transaction Service. The container is interposed between application components and J2EE services. This allows the container to transparently inject the services defined by the components' deployment descriptors, such as declarative transaction management, security checks, resource pooling, and state management.

Sun One Application Server incorporates the Web Container and the EJB Container.

This chapter includes the following topics:

- About the Web Container
- About the EJB Container

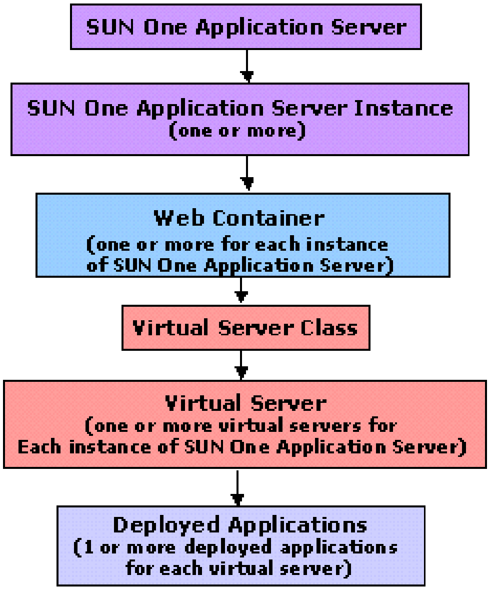
About the Web Container

The Web Container is a J2EE container that hosts web applications. The web container extends the web server functionality by providing developers the environment to run servlets and Java Server Pages (JSPs). Servlets provide a component-based, platform-independent method for building web-based applications, without the performance limitations of CGI (Common Gateway Interface) programs. The JSP technology is an extension of the servlet technology, created to support authoring of HTML and XML pages. Servlets or JSPs contained in a web container are capable of invoking bean methods in an Enterprise Java Beans (EJB) container. Bean methods are invoked either through local invocation or remote invocation using an Object Request Broker (ORB).

The web container also provides web applications access to local EJBs that are located using JNDI (Java Naming Directory Interface).

The figure "Web Container within the Sun ONE Application Server Architecture" explains the role and the location of the web container, in the Sun ONE Application Server architecture:

Figure 8-1 Web Container within the Sun ONE Application Server Architecture


SUN One Application Server
SUN One Application Server Instance
(one or more)
Web Container
(one or more for each instance
of SUN One Application Server)
Virtual Server Class
Virtual Server
(one or more virtual servers for
Each instance of SUN One Application Server)
Deployed Applications
(1 or more deployed applications
for each virtual server)

This section covers the following topics:

- Understanding the Web Container's Role
- Web Application Configuration
- Web Application Deployment
- Single Sign-on Facility
- Logging the Web Container

Understanding the Web Container's Role

The primary role of the web container is to provide a run-time environment for web applications and provide services (database access, security, multi-threading, and so on) to web applications hosted in the container. A web application is a collection of servlets, HTML pages, classes, and other resources that make up a complete application on Sun ONE Application Server.

The following are the elements of a web application:

- Servlets
- JSP pages
- Utility classes
- Static documents (html, images, sound files, etc.)
- Client side Java applets, beans, and classes
- Descriptive meta information which ties all of the above elements together.

Web applications may be deployed in the web containers running in a Sun ONE Application Server.

For more information on how to configure and use the web server plugin with Sun ONE Application Server, see Chapter 7, "Configuring the Web Server Plugin."

Web Application Configuration

You can also configure web containers to deploy web applications within virtual servers. The web container can be configured to contain more than one virtual server. Each virtual server can be configured to host any number of web applications. Web applications are scoped within the context of a virtual server. For more information about virtual servers, see Chapter 15, "Using Virtual Servers."

The following topics are covered in this section:

- Virtual Server Attributes
- Web-module Attributes

Virtual Server Attributes

You can specify values for certain configurable attributes for a virtual server. A virtual server can have more than one web application associated with it. A user needs to *sign on* to a web application.

If the attribute for single sign on, `sso-enabled`, is set to the default value true in the `server.xml` file, a user can sign on to any one of the web applications associated with the specific virtual server. The user's identity is then recognized by all other web applications running on the same virtual server. If the value for `sso-enabled` is set to false, then single sign on is disabled for all web applications in this virtual server.

The `sso-enabled` attribute is dynamically configurable and does not require a server restart to enable effect.

More information about single-sign on is provided in the section "Single Sign-on Facility" on page 206.

Web-module Attributes

The Sun ONE Application Server-specific deployment descriptors are specified in a file called `sun-web.xml`, which can be found in the `WEB-INF` directory of a given web application.

Usually, there is a `sun-web.xml` file configured for each web application. However, the web container does not require that every web application have an `sun-web.xml` file. In the absence of the `sun-web.xml` file, the web container assumes default values for all Sun ONE Application Server-specific attributes.

The context-root Attribute

This attribute defines the context root at which the web application is installed. If this attribute is an empty string then this web application is designated to be the default web application for the virtual server. The default web application for a virtual server responds to all requests that cannot be resolved to other web applications deployed to the virtual server. Every virtual server has a default web application.

For the default web application, the value of this field should be an empty string `""`.

The location Attribute

The input for this attribute should be a valid directory path, which indicates the location of the default web application. During the installation process, the location of the default web application is set to the `modules/`*default-web-app*`/` directory.

The `location` attribute is required and can be either a fully qualified or relative path to the directory in which the contents of the WAR (Web ARchive) file have been extracted. If the path specified is relative, then it needs to be relative to the *application root directory* defined at the virtual server level.

For example:

`location="`*applications/<ear name>/<war-module name>/"*

`location="`*modules/<war-module name>"*

`location="`*/u/myapps/<war-module name>"*

`location="`*/u/myapps/<ear-name>/<war-module name>"*

The enabled Attribute

The default value for this attribute is `true`, indicating that a web application is enabled to service requests. By setting the value of the enabled attribute to `false`, you can temporarily disable the web application from servicing requests. However, the contents of the web application (as stored in your hard-disk) are not removed.

Web Application Deployment

The web container deploys web applications from a Web ARchive (WAR) file or from a directory containing an exploded view (`WEB-INF/lib,WEB-INF/`classes, and so on) of the WAR file. You do not need to restart the server to deploy an application.

The web container deploys a "default" web application on each virtual server. The default location (directory) is in the `modules/`*default-web-app*`/` subdirectory of the `app root` directory for the virtual server. This default web application responds to all requests that cannot be resolved to other web applications deployed to the virtual server. This web application consists of an invoker servlet to handle requests to `/servlet/*` and a JSP servlet to serve JSP pages. The default web application can access EJBs as long the user denotes the EJB references in the `web.xml` and `sun-web.xml` files.

The default web application is defined in the `server.xml` of a virtual server looks as follows:

```
<web-module context-root="" location="modules/default-web-app/">
```

Dynamic Re-deployment and Hot Deployment

Dynamic redeployment is the ability to redeploy an existing application without a server restart. Dynamic redeployment happens when an application's configuration (contents of its `xml` files) and certain classes change. Dynamic re-deployment results in behavior identical to that of dynamic reloading the entire application's classes. In addition, dynamic re-deployment involves creating new application contexts (web and ejb) and getting rid of the old application contexts. Thus, dynamic re-deployment results in a brand new instance of the application (except for existing session data). This feature is also supported in development mode only and can result in exceptions similar to those for dynamic reloading. Also, configuration changes that require server restarts do not take effect until this restart happens. Dynamic reloading is activated only for applications and unshared standalone modules whose central configuration specifies it.

When a web application is reloaded, all existing session information is automatically saved and restored, regardless of whether a persistence mechanism was configured for the session manager.

Hot deployment is the ability to deploy an application at server runtime, without requiring a server restart. This feature uses the same infrastructure as is used for dynamic redeployment. However, since there is no state left over from a previous incarnation, this feature is supported at production time.

Single Sign-on Facility

As long as the user accesses only unprotected resources in any of the web applications on a specific virtual server, the user is not challenged to authenticate himself.

When the user accesses a protected resource in any web application associated with a specific virtual server, the user will be challenged to authenticate himself using the login method defined for the web application currently being accessed.

Once authenticated, the roles associated with the user will be utilized for access-control decisions across all of the associated web applications. The user is then not required to authenticate himself individually to each web application.

When a the user logs out of a web application the user's sessions in all web applications will be invalidated. Any subsequent attempt to access a protected resource in any application will require the user to authenticate himself 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 utilized in client environments that support cookies.

Logging the Web Container

You can control the default logging behavior of the web container and any applications that are hosted in a virtual server, by setting different *log levels*. Note that the logging behavior doesn’t affect the application’s own logging.

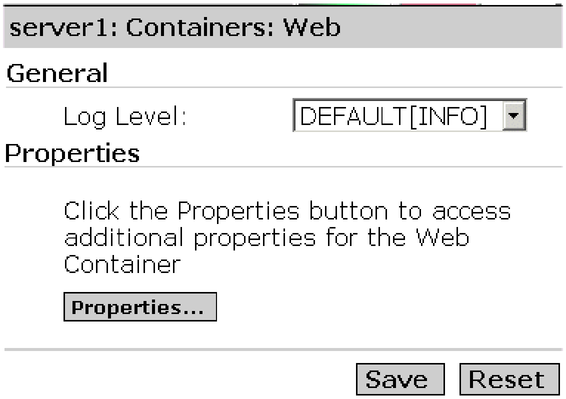
Specifying a log level controls the type of messages that will be logged. For example, if you specify that only messages that bear the log level FATAL be logged, then the messages "higher" log level than this value will be silently ignored. Only messages logged with an explicit log level are compared to this value.

Messages logged with no explicit log level are logged unconditionally. The default behavior is to log all warning, error and fatal messages.

To set log levels for the web container, perform the following tasks:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance tree to find the web container configuration you want to modify.
2. Expand the Containers tab, and select Web Container from the list of J2EE containers that are displayed. You will see the page displayed in "Logging the Web Container," in the right pane of the Administration interface.

Figure 8-2 Logging the Web Container



server1: Containers: Web

General

Log Level: DEFAULT[INFO]

Properties

Click the Properties button to access additional properties for the Web Container

3. Select the log level you want, from the Log Level drop-down list. For a listing of all the log levels and their definitions, see Chapter 5, "Using Logging."
4. Click save to save your settings

To create additional properties for the web container, click the Properties button.

About the EJB Container

Enterprise Java Bean container is a runtime environment that controls the enterprise beans and provides them with important system level services. An EJB is a component that executes within an EJB container, which in turn executes within an EJB server. The following system level services are provided to the beans:

- Transaction Management
- Security
- Life cycle Management
- Remote connectivity
- Database connection pooling
- Naming Service

Enterprise beans are server components written in Java that contain business logic. The EJB Container provides the remote access to the bean. EJBs always work within the context of a container, which serves as a link between the EJBs and the server that hosts them. The EJB container enables distributed application building using your own components and components from other suppliers.

Through the EJB container, Sun ONE Application Server provides high-level transaction, state management, multi-threading, and resource pooling wrappers, thereby shielding you from having to know the low-level API details. This container provides all standard container services denoted by the 2.0 EJB Specification, and also provides additional services specific to the Sun ONE Application Server.

Passivation and activation processes are used by the container to manage bean activity to ensure scalability.

This section covers the following topics:

- Understanding the EJB Container's Role
- Configuring the EJB Container

Understanding the EJB Container's Role

The EJB container provides the following standard services:

- Passivation

  The process of transferring an EJB from memory to secondary storage. Passivation allows a bean's resources to be released without destroying the bean. In this way, a bean is made to be persistent, and can be recalled without the overhead of instantiation.

- Activation

  The process of transferring an EJB from secondary storage to memory. The container contract establishes the relationship between an EJB and its container, and is completely transparent to a client. This relationship includes:

  - Life cycle

    For session beans, this includes the `javax.ejb.SessionBean` and `javax.ejb.SessionSynchronization` interface implementations. For entity beans, this includes the `javax.ejb.EntityBean` interface implementation. For message-driven beans, this includes the `javax.ejb.MessageDriven` interface implementation.

  - Session context

    A container implements the `javax.ejb.SessionContext` interface to pass services and information to a session bean instance when the bean instance is created.

  - Entity context

    A container implements the `javax.ejb.EntityContext` interface to pass services and information to an entity bean when the bean instance is created.

  - Message context

    A container implements the `javax.ejb.MDBContext` interface to pass services and information to a message-driven bean when the bean instance is created.

  - Environment

    A container implements `java.util.Properties` and makes these properties available to its EJBs.

  - Service information

    A container makes its services available to its EJBs.

The Sun ONE Application Server services include remote access, naming, security, concurrency, transaction control, and database access.

The EJB container is responsible for:

- Creating the implementation object (`EJBObject`) that allows remote connectivity.
- Creating the home implementation object that allows for creation of the `EJBObject`.
- Binding the home implementation object to the naming service so that clients can look up the home object.
- Ensuring that only authorized clients invoke the bean methods (through the `EJBObject`).
- Ensuring that business methods are invoked in the appropriate transactions.
- Managing the life cycle of the beans. Managing the life cycle of beans include:
    - Pooling the beans
    - Calling the appropriate callback methods (such as `ejbActivate/ejbPassivate`)
    - Managing a pool of database connections so that the applications use and reuse the connections more efficiently.

Actual implementation details are part of the container, based on a standard prescribed interface between a container and its EJBs. You are not required to know or to handle platform-specific implementation details. Instead, you can create generic, task-focused EJBs to be used with any vendor's products that support the EJB standard.

It is useful to understand the types of EJBs that are used by Sun ONE Application Server.

Types of Enterprise Java Beans

An EJB is an object that represents one of the following:

- A session with a particular client, which automatically maintains state across multiple client-invoked methods.
- A persistent entity object, possibly shared among multiple clients.
- A stateless service, such as message handling.

Entity beans are primarily used to handle data access using the Java Database Connectivity (JDBC) API, while session beans provide transient application objects and perform discrete business tasks. There are three kinds of EJBs, as discussed in the following topics:

- About Session Beans
- About Entity Beans
- About Message-driven Beans

About Session Beans

A session bean implements business rules or logic for a particular client request.

Session beans are intended to represent transient objects and processes, such as a single database record, a document copy for editing, or specialized business objects for individual clients. That is, a session bean is a private resource used only by the client that creates it. Because these objects are only available to a single client, session beans can maintain client-specific session information, called the *conversational* state.

For example, you might create an EJB to simulate an electronic shopping cart. Each time a user logs in to an application, the application creates the shopping cart session bean to hold purchases for that user. After the user logs out or finishes shopping, the session bean is released.

Session beans have the following characteristics:

- Session beans execute in relation to a single client.
- Session beans are relatively short lived.
- Session beans do not always survive a server crash.
- Session beans are removed if the EJB container crashes.
- Session beans also handle transaction management according to property settings. This is optional.
- Session beans update shared data in an underlying database. This is optional.
- A session bean can be either stateless or stateful.

Stateless Session Beans. A stateless session bean encapsulates a temporary piece of business logic needed by a specific client for a limited time span. Stateless session beans do not maintain the conversational state.

Stateful Session Beans. A stateful session bean is transient, but maintains a conversational state to preserve information about its contents and values between client calls. The conversational state enables the bean's container to maintain information about the session bean state and to recreate the state at a later point in program execution when needed.

About Entity Beans

Entity beans commonly represent persistent data which is maintained directly in a database or accessed through an Enterprise Information System (EIS) application as an object. The server that hosts EJBs and an EJB container provides a scalable runtime environment for concurrently active entity EJBs.

A simple example of an entity bean is one defined to represent a single row in a database table, where each bean instance represents a specific row. A more complex example is an entity bean designed to represent complicated views of joined tables in a database, where, for example, each bean instance represents the contents of a single shopping cart.

Entity beans have the following characteristics:

- Entity beans provide an object view of data in the EIS resource, usually a database.
- Entity beans can be accessed by all users.
- Entity beans transparently survive server crashes.
- Entity beans use transactions that are either container-managed or bean-managed.

Entity beans represent persistent data, either as container-managed persistence or bean-managed persistence. An entity bean's persistence can either be managed by the bean or the container.

Bean-managed persistence. When an entity bean manages its own persistence. The bean developer implements persistence code (such as JDBC calls) directly in the EJB class methods. The possible downside is portability loss if a proprietary interface is used, and the risk in associating the bean to a specific database.

Container-managed persistence. When entity bean persistence is managed by the container. Because the container transparently manages the persistence state, you do not need to implement any data access code in the bean methods. Not only is this method simpler to implement, but it makes the bean fully portable without any ties to a specific database.

An entity bean that uses container-managed persistence is essentially an auto-generated (by the container) version of an entity bean that uses bean-managed persistence.

For more information on building and using Entity beans, see the *Sun ONE Application Server Developer’s Guide to Enterprise JavaBeans Technology*.

About Message-driven Beans

A message-driven bean is an EJB that allows J2EE applications to process messages asynchronously. A message-driven bean is driven by the arrival of Java Message Service messages.

From its creation until destruction, a message-driven bean instance lives in a message-driven bean container. The container provides security, transaction, concurrent processing of messages, life cycle management of the message-driven bean instances, and other services for the message-driven bean. The server that hosts EJBs and an EJB container provides a scalable runtime environment for concurrently active message-driven beans.

The Java Message Service API in the J2EE 1.3 platform specifies the following:

- Application clients, EJB components, and web components can send or synchronously receive a Java Message Service message. In addition, application clients can use Java Message Service messages asynchronously.
- The message-driven beans enable the asynchronous consumption of messages. A Java Message Service provider may optionally implement concurrent processing of messages by message-driven beans.

A message-driven bean represents a stateless service; it is essentially an asynchronous message consumer that is completely anonymous and has no client-visible identity. A message-driven bean has neither a home nor a component interface. A client accesses a message-driven bean through Java Message Service by sending messages to the Java Message Service destination (queue or topic) for which the message-driven bean class is the MessageListener.

Only message-driven beans can asynchronously receive messages. A session or entity beans is not permitted to be a Java Message Service `MessageListener`.

Message-driven beans have the following characteristics:

- Execute upon receipt of a single client message.
- Are asynchronously invoked.
- Are relatively short lived.
- Do not directly represent shared data in the database, but may access and update this data.
- Are removed if the EJB server crashes.
- Are stateless.
- Optionally, are transaction-aware.

Configuring the EJB Container

You can configure the log level for the EJB container, and also enable monitoring. The EJB container handles both EJBs and MDBs. Using the Administration interface, you can configure settings for the EJBs and the MDBs that the container manages. This section covers the following topics:

- Performing General Configuration
- Configuring EJB Settings
- Configuring MDB Pool Settings

Performing General Configuration

You configure the following aspects of the EJB container:

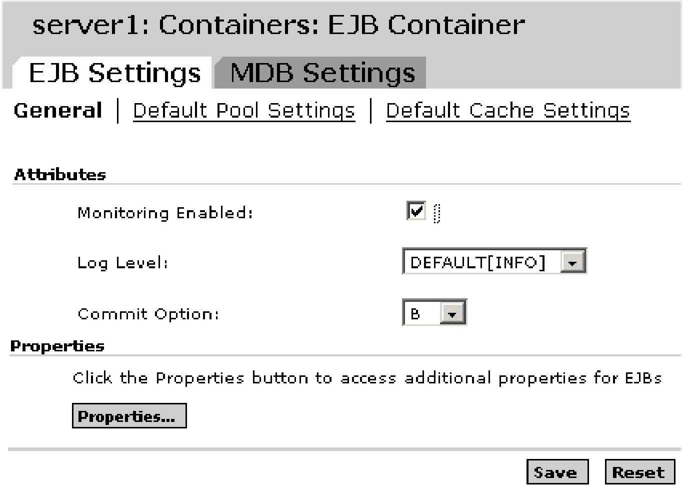
- Logging
- Monitoring
- Transaction Attributes

To set log levels for the EJB container, to enable monitoring and to set transaction attributes, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance tree, for the EJB container configuration you want to modify.

2. Expand the Containers tab, and select EJB Container from the list of J2EE containers that are displayed. You will see the window as shown in “EJB Container - General Configuration” in the right pane of the Administration interface.

Figure 8-3 EJB Container - General Configuration



Monitoring Enabled: ☑

Log Level: DEFAULT[INFO]

Commit Option: B

Click the Properties button to access additional properties for EJBs

3. Mark the checkbox against Monitoring Enabled, to enable monitoring of the EJB Container. You've now enabled monitoring for the EJB container of this specific instance of Sun ONE Application Server. For a list of the monitorable aspects of the EJB container, see the table "Monitoring Statistics for EJB Container."

4. Select the log level you want, from the Log Level drop-down list. For a listing of all the log levels and their definitions, Chapter 5, "Using Logging." Specifying a log level controls the type of messages that will be logged. For example, if you specify that only messages that bear the log level FATAL be logged, then the messages "higher" log level than this value will be silently ignored. Only messages logged with an explicit log level are compared to this value.

   Messages logged with no explicit log level are logged unconditionally. The default behavior is to log all warning, error and fatal messages.

5. From the Commit Option drop down list, select the Commit Option you want to use for the EJB Container.

   A transaction can end in two ways: with a commit or a rollback. When a transaction commits, the data modifications made by its statements are saved. When you design an enterprise bean, you determine if the commit is a container-managed or a bean-managed transaction. The options in the UI are therefore, B for bean-managed commit, and C for container-managed commit.

6. Click the Properties button to create new properties for the EJB container.

7. Click OK to save your settings.

The following table displays the attributes of the EJB container that can be monitored:

Table 8-1 Monitoring Statistics for EJB Container

Statistic Name	Data Type & units	Range of values	Comments
minBeansInPool	Integer	0-MAXINT	The preferred minimum number of beans in the pool (applies to stateless session beans
initialBeansInPool	Integer	0-MAXINT	The initial number of beans in the pool (applies to stateless session beans).
maxBeansInPool	Integer	0-MAXINT	Maximum number of beans in the pool. (applies to stateless session beans).
beanIdleTimeoutInSeconds	Integer	0-MAXLONG	Idle timeout in seconds beyond which the bean will be destroyed.
numBeansCreated	Integer	0-MAXINT	Number of beans so far created.
numBeansDestroyed	Integer	0-MAXINT	Number of beans so far destroyed.
numThreadsWaitaing	Integer	0-MAXINT	Number of threads waiting for free beans
numBeansInPool	Integer	0-MAXINT	Number of beans available in pool. (If this is greater than zero, then numThreadsWaitaing must be 0)
maxBeansInCache	Integer	0-MAXINT	The maximum number of beans in the cache (applies to entity and stateful beans).
minBeansInCache	Integer	0-MAXINT	The preferred minimum number of beans in the cache (applies to entity and stateful beans).
cacheFaultsPercentage	Double		Number of cache misses that resulted in activating from backup store.

Configuring EJB Settings

Using the Administration interface, you can configure the default pool and bean cache settings for the EJBs managed by the EJB container, as discussed in the following topics:

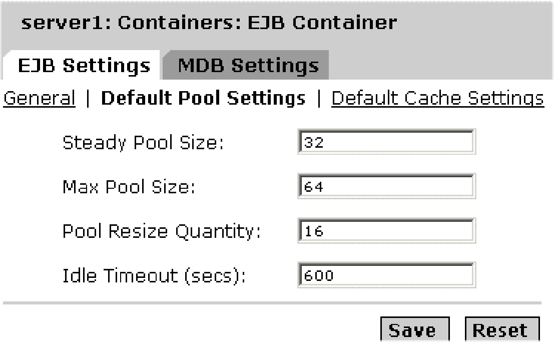
- To Configure EJB Pool Settings
- To Configure EJB Cache Settings

To Configure EJB Pool Settings

To configure EJB pool settings, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance tree, whose EJB settings you want to modify.
2. Expand the Containers tab, and select EJB Container from the list of J2EE containers that are displayed. You will see the window as shown in "Configuring EJB Pool Settings" in the right pane of the Administration interface.

Figure 8-4 Configuring EJB Pool Settings



server1: Containers: EJB Container

Steady Pool Size: 32

Max Pool Size: 64

Pool Resize Quantity: 16

Idle Timeout (secs): 600

3. In the Steady Pool Size field, specify the minimum number of beans in the pool. This applies to stateless session beans.
4. In the Max Pool Size drop down list, specify the maximum number of beans you want in the pool, at any given point in time. This setting applies to stateless session beans
5. In the Pool Resize Quantity field, specify the number of beans to be removed from the pool, if the beans are idle for more than the time specified in the idle-timeout-in-seconds tag.
6. In the Idle Timeout (secs) field, specify the period, in seconds, that a bean can remain idle. When the idle-timeout period elapses and the bean is still idle, it will be destroyed.
7. Click Save to save your changes.

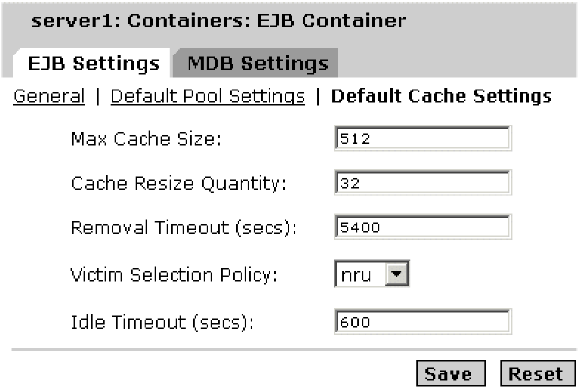
To Configure EJB Cache Settings

To configure EJB cache settings, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance tree, whose EJB settings you want to modify.

2. Expand the Containers tab, and select EJB Container from the list of J2EE containers that are displayed. You will see the window as shown in "Configuring EJB Pool Settings" in the right pane of the Administration interface.

Figure 8-5 Configuring EJB Cache Settings


server1: Containers: EJB Container
EJB Settings
MDB Settings
General | Default Pool Settings | Default Cache Settings
Save
Reset

Max Cache Size: 512

Cache Resize Quantity: 32

Removal Timeout (secs): 5400

Victim Selection Policy: nru

Idle Timeout (secs): 600

3. In the `Max Cache Size` field, specify the maximum number of beans to be maintained in the cache. The default value for this attribute is as specified in the `idle-timeout-in-seconds` attribute.

4. In the `Cache Resize Quantity` field, specify the number of beans to be selected for destruction, if the number of beans in the pool exceeds the quantity specified in the `Max Cache Size` attribute.

5. In the `Removal Timeout` (secs) field, specify the amount of time that a bean that is idle in the backup store can remain passivated. If a bean is not accessed by a client beyond the value specified in the `removal-timeout-in-seconds` attribute, then the bean will be removed from the backup store and therefore will not be accessible to the client.

6. From the `Victim Selection Policy` drop-down list, select the victim selection algorithm that must be employed to select victim beans to be removed from the pool.

7. In the `Idle Timeout` (secs) field, specify the period for which a bean is allowed to be idle in the cache. After this period elapses, the bean is passivated. The period for which the bean remains passivated (in the idle backup store), is controlled by the `removal-timeout-in-seconds` parameter.

8. Click Save to save your changes.

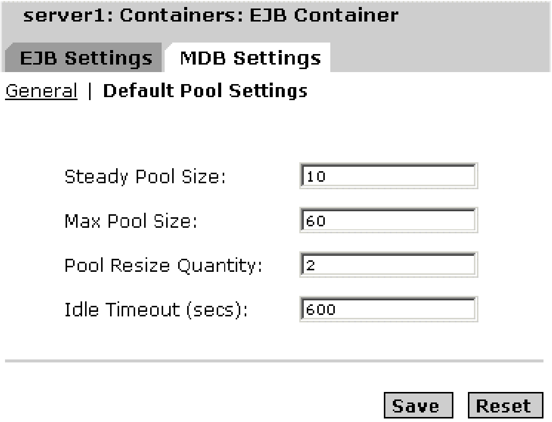
Configuring MDB Pool Settings

Using the Administration interface, you can configure the default pool settings for the MDBs managed by the EJB container. To configure default pool settings for MDBs, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance tree, whose MDB container configuration you want to modify.

2. Expand the Containers tab, and select EJB Container from the list of J2EE containers that are displayed. You will see the window shown in "Configuring MDB Pool Settings" in the right pane of the Administration interface.

Figure 8-6 Configuring MDB Pool Settings



server1: Containers: EJB Container

General | Default Pool Settings

Steady Pool Size: 10

Max Pool Size: 60

Pool Resize Quantity: 2

Idle Timeout (secs): 600

3. Click MDB Settings. In the `Steady Pool Size` text field, specify the minimum number of beans in the pool. This applies to stateless session beans.

4. In the `Max Pool Size` field specify the maximum number of beans you want in the pool, at any given point in time.

5. In the Pool Resize Quantity field, specify the number of beans to be removed from the pool, if the beans are idle for more than the time specified in the `idle-timeout-in-seconds` tag.

6. In the `Idle Timeout` (secs) field, specify the period, in seconds, that a bean can remain idle. When the idle-timeout period elapses and the bean is still idle, it will be destroyed.

7. Click Save to save your settings.

Chapter 9

Using Transaction Services

Transactions are an integral part of business. A typical business transaction involves transfer of assets between two or more involved parties. Accurate records are usually stored in one or more databases. Because this information is critical for business operations, it must be valid, current, and reliable. Transaction processing can be difficult to a novice programmer. The J2EE platform provides several abstractions that simplify development of dependable transaction processing applications. In this chapter, we will discuss J2EE transactions and transaction support in Sun ONE Application Server.

This chapter discusses Java Transactions in general, and the transaction support incorporated into Sun ONE Application Server in specific.

This chapter includes the following topics:

- What Is a Transaction?
- Transactions in J2EE
- Transactional Resource Managers
- Local and Distributed Transactions
- Container-Managed Transactions
- Bean-Managed Transactions
- Transaction Service Administration

What Is a Transaction?

To emulate a business transaction, a program may need to perform several steps. A financial program, for example, might transfer funds from a checking account to a savings account, by performing the steps listed in the following pseudocode:

```
begin transaction
debit checking account
credit savings account
update history log
commit transaction
```

In the preceding pseudocode, the `begin` and `commit` statements mark boundaries of the transaction. To complete this transaction, all the three steps complete must complete successfully. If all three steps don't complete successfully, data integrity could be compromised.

This guarantee is described as `atomicity`. A transaction can end in two ways: with a `commit` or a `rollback`. When a transaction commits, the modifications made by statements within the transaction boundaries are saved and made permanent. The changes are *durable*, that is they will survive future system failures. If any statement within a transaction fails, the transaction rolls back, undoing the effects of all statements executed so far in the transaction. In the pseudocode, for example, if a disk drive crashed during the credit step, the transaction would roll back and undo the data modifications made by the debit statement.

Even if a transaction fails, data integrity would be intact because the transaction accounts still balance. This aspect of transactional behavior is known as transactional *consistency*.

The transaction service also provides *isolation*, which means that phases in a transaction cannot be observed by other applications and threads, until the transaction is committed or rolled back. Once a transaction is committed, the committed transaction can be safely observed by applications and threads.

Transactions in J2EE

Transaction processing in J2EE involves the following five participants: Transaction Manager, Application Server, Resource Manager(s), Resource Adapter(s) and the User Application. Each of these entities contribute to reliable transaction processing, by implementing different APIs and functionalities, discussed below:

- Transaction Manager provides the services and management function required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.

- The Application server provides the infrastructure required to support the application run-time environment which includes transaction state management.
- Resource Manager (through a resource adapter) provides the application access to resources. The resource manager participates in distributed transactions by implementing a transaction resource interface used by the transaction manager to communicate transaction association, transaction completion and recovery work. An example of such a resource manager is a relational database server.
- A Resource Adapter is a system level software library that is used by the application server or client to connect to a Resource Manager. A Resource Adapter is typically specific to Resource Manager. It is available as a library and is used within the address space of the client using it. An example of such a resource adapter is the JDBC driver.
- A transactional User Application developed to operate in a J2EE application server environment, looks up transactional data sources, and optionally, the transaction manager, using JNDI. May use declarative transaction attribute settings for EJBs or explicit programmatic transaction demarcation.

The term Resource Manager is often used interchangeably with Resource Adapter, as there is a close tie between the two entities.

Transactional Resource Managers

The following transactional resource managers are supported in J2EE transactions.

- Databases
- JMS Providers
- J2EE Connectors

Databases

Databases are the most commonly encountered transactional resource managers in J2EE applications. JDBC is the API is used by J2EE components to access databases. Database resources are configured as JDBC resources. JDBC resources are managed by a resource managers, or JDBC Drivers. A JDBC driver may provide support for Local transactions or Global transactions, and in some cases for both local and global transactions.

Sun ONE Application Server supports usage of JDBC and Transactions from various J2EE components. For more details on how JDBC resources are registered and configured, see "About JDBC Resources" on page 264. The Application Server is responsible for providing transaction continuity (that is, initiating a transaction and accessing the database from multiple application components). For example, a servlet may start a transaction, access a database, invoke an enterprise bean that accesses the same database as part of the same transaction, and finally, commit the transaction.

JMS Providers

JMS stands for Java Message Service. A JMS Provider is the J2EE term for the Message Broker Service. The JMS API provides reliable and transactional exchange of messages between applications. Support of transactional JMS data sources is a required capability in J2EE. JMS resources and JDBC resources can participate in the same transaction.

Sun ONE Application Server comes integrated with Sun ONE Message Queue, a fully capable JMS provider and the corresponding transactional resource manager. In this manner, Sun ONE Application Server enables transactional JMS access from servlets, JSP pages and enterprise beans. It is also possible to use third party JMS Providers with Sun ONE Application Server. For more details, see Chapter 11, "Using the JMS Service."

J2EE Connectors

Sun ONE Application Server supports resource adapters that use XATransaction mode as transaction resource managers. The platform must enable transactional access to the resource adapter from servlets, JSP pages, and enterprise beans. It is possible to access the resource adapter from multiple application components within a single transaction. For example, a servlet may wish to start a transaction, access the resource adapter, invoke an enterprise bean that also accesses the resource adapter as part of the same transaction, and finally, commit the transaction.

Local and Distributed Transactions

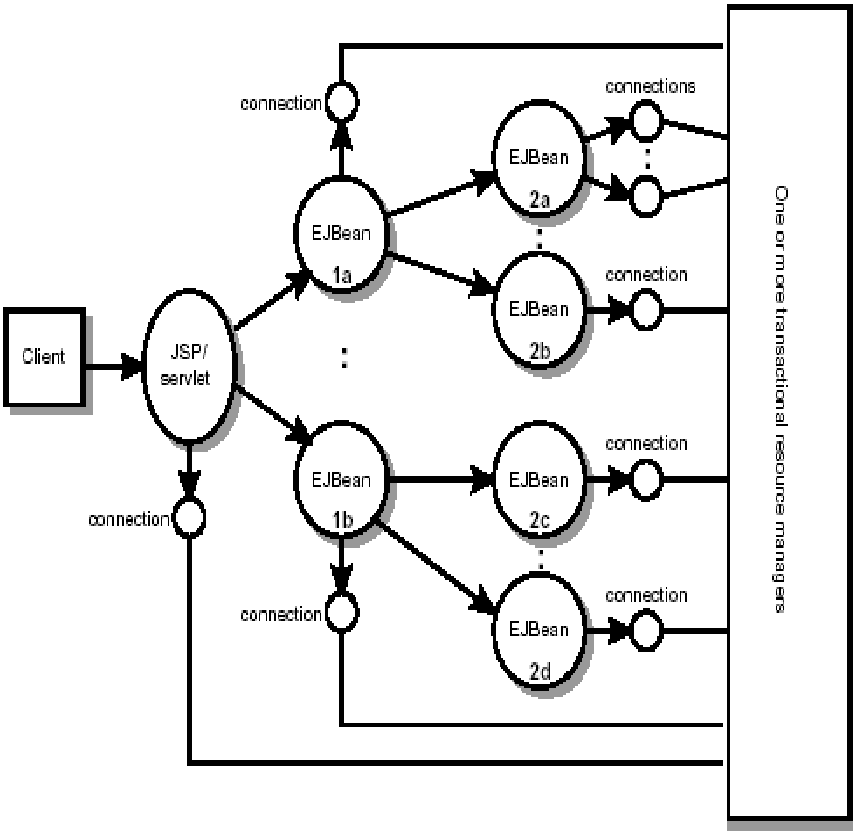
A transaction that involves only one resource can be completed using local transactions. A local transaction also requires that all participating application components execute within one process. Transactions that involve more than one resource, or multiple participant processes become distributed or global transactions. Local transaction optimization uses the resource manager specific optimization and it is transparent to the J2EE application.

The type of transaction is largely determined by the interfaces implemented of the involved resource managers. For example, a JDBC Data Source that implements `javax.sql.DataSource` interface can participate in Local transactions. A Data Source that implements `javax.sql.XADataSource` is capable of taking part in global transactions. Some JDBC resources implement both the interfaces, and when such a JDBC resource is registered with Sun ONE Application Server, it may be necessary to provide additional configuration information in the Sun ONE Application Server configuration to indicate the preferred capability for that resource.

Local transactions are simpler and naturally more efficient than global transactions. Local transactions are inadequate when the data that needs to be transformed is spread across multiple data sources. Sometimes, it is not possible to predict how many data sources would need to be enlisted in a transaction. So global transactions are encountered quite frequently in the real world. Some performance enhancing optimizations are possible with global transactions.

J2EE supports transactional applications comprising any combination of servlets or JSPs accessing multiple enterprise beans within a transaction. Each component may acquire one or more connections to access one or more transactional resource managers. In the following figure, the call tree starts from a servlet or JSP page accessing multiple enterprise beans, which in turn may access other enterprise beans. The components access resource managers via connections.

J2EE Components Accessing Resources in a Transaction



For example, an application may require that all the components in the above figure access resources as part of a single transaction. The application server provider must provide the transaction capabilities to support such a scenario.

J2EE transactional management supports flat transactions. A flat transaction cannot have any child (nested) transactions.

Transaction Recovery is an important aspect of distributed transactions. When a resource becomes unreachable during critical points, or if there are other unrecoverable errors, the status of the distributed transaction can be in question. Automatic and manual recovery of stranded/incomplete transactions is an important feature in Sun ONE Application Server. You can enable automatic transaction recovery by using the Administration interface. For more information on how to control transaction recovery, see "Transaction Service Administration" on page 235.

Connections—used as a synonym here for resources—can be marked as either shareable or non-shareable. A J2EE application component that intends to use a connection in an un-shareable way must provide deployment information to that effect, to prevent the connection from being shared by the container. Examples of when this may be needed include situations with changed security attributes, isolation levels, character settings, and localization configuration.

Containers must not attempt to share connections that are marked un-shareable. If a connection is not marked as unshareable, it must be transparent to the application whether the connection is actually shared or not.

J2EE application components may use the optional deployment descriptor element `res-sharing-scope` to indicate whether a connection to a resource manager is shareable or unshareable. Containers should assume connections to be shareable if no deployment hint is provided. J2EE application components may cache connection objects and reuse them across multiple transactions. Containers that provide connection sharing should transparently switch such cached connection objects (at dispatch time) to point to an appropriate shared connection with the correct transaction scope.

When designing an enterprise bean application, the developer must determine how the boundaries are specified.

Container-Managed Transactions

In an enterprise bean with container-managed transactions, the EJB container sets the boundaries of the transactions. You can use container-managed transactions with any type of enterprise bean: session, entity, or message-driven. Container-managed transactions simplify development because the enterprise bean code does not explicitly mark the transaction's boundaries. The code does not include statements that begin and end the transaction.

Typically, the container begins a transaction immediately before an enterprise bean method starts. It commits the transaction just before the method exits. Each method can be associated with a single transaction. Nested or multiple transactions are not allowed within a method.

Container-managed transactions do not require all methods to be associated with transactions. When deploying a bean, you specify which of the bean's methods are associated with transactions by setting the transaction attributes.

This section covers the following topics:

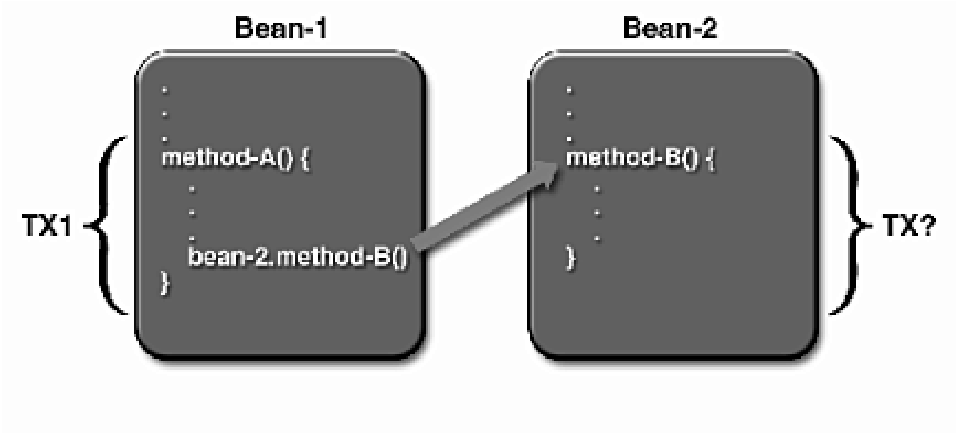
- Transaction Attributes

- Setting Transaction Attributes
- Rolling Back a Container-Managed Transaction
- Synchronizing a Session Bean's Instance Variables
- Methods Not Allowed in Container-Managed Transactions

Transaction Attributes

A transaction attribute controls the scope of a transaction. The following figure illustrates why controlling the scope is important. In the diagram, method-A begins a transaction and then invokes method-B of Bean-2. When method-B executes, does it run within the scope of the transaction started by method-A or does it execute with a new transaction? The answer depends on the transaction attribute of method-B.

Figure 9-1 Transaction Attributes



A transaction attribute may have one of the following values:

- Required
- RequiresNew
- Mandatory
- NotSupported
- Supports

- Never

Required

If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction. If the client is not associated with a transaction, the container starts a new transaction before running the method.

The `Required` attribute will work for most transactions. Therefore, you may want to use it as a default, at least in the early phases of development. Because transaction attributes are declarative, you can easily change them at a later time.

RequiresNew

If the client is running within a transaction and invokes the enterprise bean's method, the container takes the following steps:

- Suspends the client's transaction
- Starts a new transaction
- Delegates the call to the method
- Resumes the client's transaction after the method completes

If the client is not associated with a transaction, the container starts a new transaction before running the method.

You should use the `RequiresNew` attribute when you want to ensure that the method always runs within a new transaction.

Mandatory

If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction. If the client is not associated with a transaction, the container throws the `TransactionRequiredException`.

Use the `Mandatory` attribute if the enterprise bean's method must use the transaction of the client.

NotSupported

If the client is running within a transaction and invokes the enterprise bean's method, the container suspends the client's transaction before invoking the method. After the method has completed, the container resumes the client's transaction.

If the client is not associated with a transaction, the container does not start a new transaction before running the method.

Use the NotSupported attribute for methods that don't need transactions. Because transactions involve overhead, this attribute may improve performance.

Supports

If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction. If the client is not associated with a transaction, the container does not start a new transaction before running the method.

Because the transactional behavior of the method may vary, you should use the Supports attribute with caution.

Never

If the client is running within a transaction and invokes the enterprise bean's method, the container throws a RemoteException. If the client is not associated with a transaction, the container does not start a new transaction before running the method.

Attribute Summary

The following table summarizes the effects of the transaction attributes. Both the T1 and T2 transactions are controlled by the container. A T1 transaction is associated with the client that calls a method in the enterprise bean. In most cases, the client is another enterprise bean. A T2 transaction is started by the container just before the method executes.

In the last column of, the term "None" indicates that the business method does not execute within a transaction controlled by the container. However, the database calls in such a business method might be controlled by the transaction manager of the DBMS.

Table 9-1 Transaction Attributes

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2

Table 9-1 Transaction Attributes

Mandatory	None	error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1

Setting Transaction Attributes

Because transaction attributes are stored in the deployment descriptor, they can be changed during several phases of J2EE application development: enterprise bean creation, application assembly, and deployment. However it is the responsibility of a developer to specify the attributes when creating the bean. The attributes should be modified only by an application developer who is assembling components into larger applications. The individual who deploys the J2EE application is not responsible for specifying the transaction attributes.

You can specify the transaction attributes for the entire enterprise bean or for individual methods. If you've specified one attribute for a method and another for the bean, the attribute for the method takes precedence. When specifying attributes for individual methods, the requirements differ with the type of bean. Session beans need the attributes defined for business methods, but do not allow them for the create methods. Entity beans require transaction attributes for the business, create, remove, and finder methods. Message-driven beans require transaction attributes (either `Required` or `NotSupported`) for the `onMessage` method.

Rolling Back a Container-Managed Transaction

There are two ways to roll back a container-managed transaction. First, if a system exception is thrown, the container will automatically roll back the transaction. Second, by invoking the `setRollbackOnly` method of the `EJBContext` interface, the bean method instructs the container to roll back the transaction. If the bean throws an application exception, the rollback is not automatic, but may be initiated by a call to `setRollbackOnly`.

In the following example, the transferToSaving method of the BankEJB example illustrates the setRollbackOnly method. If a negative checking balance occurs, transferToSaving invokes setRollBackOnly and throws an application exception (InsufficientBalanceException). The updateChecking and updateSaving methods update database tables. If the updates fail, these methods throw a SQLException and the transferToSaving method throws an EJBException. Because the EJBException is a system exception, it causes the container to automatically roll back the transaction. Here is the code for the transferToSaving method:

```
public void transferToSaving(double amount) throws
   InsufficientBalanceException  {

   checkingBalance -= amount;
   savingBalance += amount;

if (checkingBalance < 0.00) {
context.setRollbackOnly();

throw new InsufficientBalanceException();
}
   try {
      updateChecking(checkingBalance);

   updateSaving(savingBalance);
   } catch (SQLException ex) {
      throw new EJBException
         ("Transaction failed due to SQLException: "
         + ex.getMessage());
   }
}
```

When the container rolls back a transaction, it always undoes the changes to data made by SQL calls within the transaction. However, only in entity beans will the container undo changes made to instance variables. (It does so by automatically invoking the entity bean’s ejbLoad method, which loads the instance variables from the database.) When a rollback occurs, a session bean must explicitly reset any instance variables changed within the transaction. The easiest way to reset a session bean’s instance variables is by implementing the SessionSynchronization interface.

You can also roll back a transaction by passing the transaction ID through the command line interface. For more details, please see “Administering Transactions Using the Command-Line Interface” on page 238.

Synchronizing a Session Bean's Instance Variables

The SessionSynchronization interface, which is optional, allows you to synchronize the instance variables with their corresponding values in the database. The container invokes the SessionSynchronization methods—afterBegin, beforeCompletion, and afterCompletion—at each of the main stages of a transaction.

The afterBegin method informs the instance that a new transaction has begun. The container invokes afterBegin before it invokes the first business method within a transaction. The afterBegin method is a good place to load the instance variables from the database. The BankBean class, for example, loads the checkingBalance and savingBalance variables in the afterBegin method:

```
public void afterBegin() {

   System.out.println("afterBegin()");
   try {
      checkingBalance = selectChecking();
      savingBalance = selectSaving();
   } catch (SQLException ex) {
      throw new EJBException("afterBegin Exception: " +
         ex.getMessage());
   }
}
```

The container invokes the beforeCompletion method after the business method has finished, but just before the transaction commits. The beforeCompletion method is the last opportunity for the session bean to roll back the transaction (by calling setRollbackOnly). If it hasn't already updated the database with the values of the instance variables, the session bean may do so in the beforeCompletion method.

The afterCompletion method indicates that the transaction has completed. It has a single boolean parameter, whose value is true if the transaction was committed and false if it was rolled back. If a rollback occurred, the session bean can refresh its instance variables from the database in the afterCompletion method:

```
public void afterCompletion(boolean committed) {

   System.out.println("afterCompletion: " + committed);
   if (committed == false) {
      try {
         checkingBalance = selectChecking();
         savingBalance = selectSaving();
} catch (SQLException ex) {
          throw new EJBException("afterCompletion SQLException:
          " + ex.getMessage());
}
}
}
```

Methods Not Allowed in Container-Managed Transactions

You should not invoke any method that might interfere with the transaction boundaries set by the container. The list of prohibited methods follows:

- The commit, `setAutoCommit`, and rollback methods of `java.sql.Connection`
- The `getUserTransaction` method of `javax.ejb.EJBContext`
- Any method of `javax.transaction.UserTransaction`

You may, however, use these methods to set boundaries in bean-managed transactions.

Bean-Managed Transactions

In a bean-managed transaction, the code in the session or message-driven bean explicitly marks the boundaries of the transaction. An entity bean cannot have bean-managed transactions; it must use container-managed transactions instead. Although beans with container-managed transactions require less coding, they have one limitation: When a method is executing, it can be associated with either a single transaction or no transaction at all. If this limitation will make coding your bean difficult, you should consider using bean-managed transactions.

The following pseudocode illustrates the kind of fine-grained control you can obtain with bean-managed transactions. By checking various conditions, the pseudocode decides whether to start or stop different transactions within the business method.

```
begin transaction
...
update table-a
...
if (condition-x)
   commit transaction
```

```
else if (condition-y)
   update table-b
   commit transaction
else
   rollback transaction
   begin transaction
   update table-c
   commit transaction
```

Transaction Service Administration

You can administer transaction by using the Administration interface, or by using the Command Line Interface.

This section covers the following topics:

- Administering Transactions Using the Administration Interface
- Administering Transactions Using the Command-Line Interface

Administering Transactions Using the Administration Interface

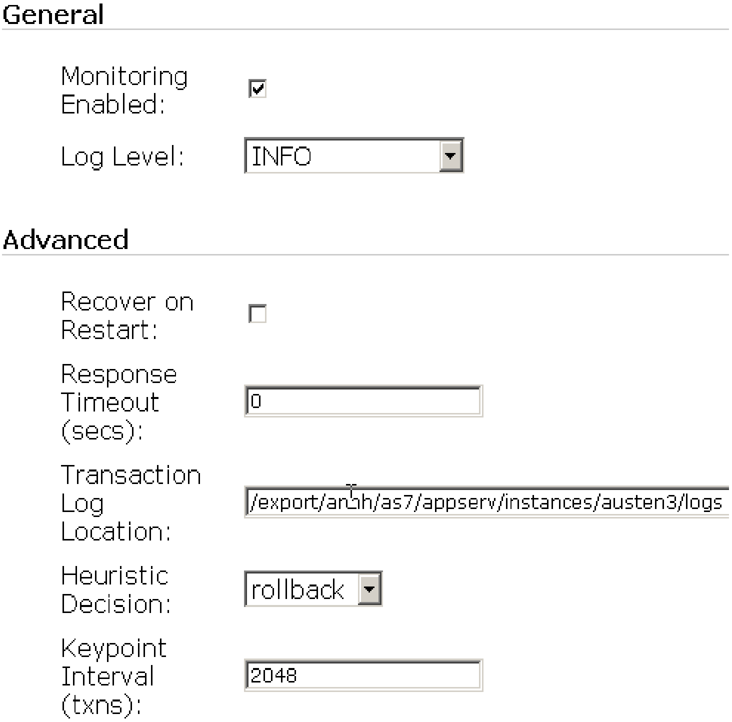
Using the Administration interface, you can enable monitoring, set log levels, and specify advanced options for your transactions.

You can control instance-wide transaction service attributes, such as recovery policy and time-outs. The properties and configuration that you specify here are stored in the `server.xml` file.

To configure transaction service options, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance tree, whose transaction configuration you want to modify.
2. Select Transaction Service from the list of J2EE services that are displayed. You will see the following window, shown in the figure “Configuring Transaction Service Options” in the right pane of the Administration interface:

Figure 9-2 Configuring Transaction Service Options



General

Monitoring Enabled: ☑

Log Level: INFO

Advanced

Recover on Restart: □

Response Timeout (secs): 0

Transaction Log Location: /export/an[illegible]h/as7/appserv/instances/austen3/logs

Heuristic Decision: rollback

Keypoint Interval (txns): 2048

3. To enable monitoring for your transactions, mark the Monitoring Enabled checkbox. The following table lists the features of Java Transaction services that can be monitored:

Table 9-2 Monitorable Attributes of Java Transaction Services

Property	Type	Description
transactionsCompleted	int	Number of transactions completed since monitoring is enabled
transactionsRolledBack	int	Number of transactions rolled back since monitoring is enabled
transactionsRecovered	int	Number of transactions recovered since monitoring is enabled

Table 9-2 Monitorable Attributes of Java Transaction Services

transactionsInFlight	int	Number of transactions presently being processed
timeStamp	long	In milliseconds, recording the time at which the statistic was produced. This will be whatever that is reported by `System.getCurrentTimeInMillis()`

4. Select the log level you want to set for your transactions, from the Log Level drop-down list. For a more information on log levels and how they are incorporated, see Chapter 5, “Using Logging.”

5. Mark the checkbox against Recover on Restart to automatically recover failed transactions at server restart. When a resource becomes unreachable during critical points in the transaction Commit protocol, transactions may not complete and remain in transaction log file. If this check box has been marked, the server attempts to recover stranded transaction upon server restart. If the involved resources remain unreachable, this may delay server restart. This checkbox is not marked by default.

6. For enterprise beans with container-managed transactions, you can control the transaction timeout interval by setting the value of the Transaction Timeout (secs) property.

   If the value of this property is set to 0, your transaction does not time out.

   In the Transaction Timeout (secs) field, specify the transaction timeout interval. If a transaction is not completed within the specified time, the transaction will be rolled back. If the value set for this attribute is 0, the transaction does not time out.

7. In the Transaction Log Location field, specify the absolute path to the directory in which you want to store your log files. You need to restart your server for the new transaction log directory to be effective.

8. From the Heuristic Decision drop-down box, select the heuristic decision that you want to apply to your transactions. Select Commit or Rollback from the options indicated, to specify how the Application Server should determine the outcome of an in-doubt transaction during recovery, if the outcome cannot be determined unambiguously. If the Heuristic Decision is set to Rollback, it rolls back the transaction. In some cases, it may be acceptable to commit such a transaction.

9. In the Keypoint Interval (transactions) field, specify the number of transactions between keypoint operations in the log. Keypoint operations reduce the size of the transaction log file by removing entries for completed transactions and compressing the file. A larger value for this attribute results in a larger transaction log file, but fewer keypoint operations mean a potentially better performance. A smaller value (for example, 100) results in smaller log files, but slightly reduced performance due to the greater frequency of keypoint operations.

Administering Transactions Using the Command-Line Interface

You can administer and monitor database transactions using the Command Line Interface (CLI), as explained in the following sections:

- Listing In-Flight Transactions
- Managing Transactions
- Freezing the Transaction Service
- Monitoring Transactions

These sections explain how to use the Command Line Interface to manage and monitor transactions.

Listing In-Flight Transactions

The following command should be used to get the in-flight transaction data (assuming that you are in multimode and have already set the user name and password):

```
- asadmin> get --monitor
<instanceName>.transaction-service.inflight-tx
```

The multi-line output will look like this:-

```
Transaction Id State Elapsed Time (ms)

txnid1 Prepared 20

txnid2 Active 100

txnid3 Active 120

  ... ... ...
```

Managing Transactions

In the example given in Listing In-Flight Transactions, let us assume that you want to rollback transactions with the transaction Ids: `txn-ids`, `txnid2`, and `txnid3`. A sample command to rollback chosen transactions, would look like the following example:

```
asadmin> set --monitor
<instanceName>.transaction-service.rollback-list=txnid2,txnid3
```

Freezing the Transaction Service

To freeze the transaction service, run the following command:

```
asadmin> set --monitor
<instanceName>.transaction-service.freeze=true
```

When the transaction service is frozen the transaction manager in the application server, will suspend all in-flight transactions. Freezing is not recommended on a production deployment system.

To unfreeze the transaction service, run the following command:

```
asadmin> set --monitor
<instanceName>.transaction-service.freeze=false
```

When the transaction service is set in motion again, the system continues where it left off. If a live system was left in frozen state for too long, some database connections may time out, resulting in rolled back transactions.

Monitoring Transactions

To get the monitoring data of transactions, including in-flight transaction data, run the following command:

```
asadmin> get --monitor <instanceName>.transaction-service.*
```

If there are no active transactions when you run this command, you will get the following output:

```
total-tx-completed = 5

total-tx-rolledback = 2

total-tx-inflight = 0

isFrozen = false

tx-inflight = No active transactions found.
```

If there are active transactions, when you run this command, you will get the following output:

```
total-tx-completed = 5

total-tx-rolledback = 2

total-tx-inflight = 2

isFrozen = false

tx-inflight =

Transaction Id State Elapsed Time(ms)

txnid1 Prepared 500

txnid2 Active 360
```

Chapter 10

Configuring Naming and Resources

This chapter describes the J2EE Resources used by Sun ONE Application Server and discusses the methods used to create and manage these resources:

This chapter includes the following topics:

About J2EE Naming Services and Resources

J2EE applications including EJBs, web application components and application clients may access a wide variety of resources such as resource managers, data sources (for example SQL datasources), connection factories, mail sessions, Java Message Service destination objects, and URL connection factories. The J2EE platform exposes such resources to the applications via Java Naming and Directory (JNDI) naming service.

Sun ONE Application Server allows you to create and manage the following J2EE resources:

- JDBC Datasources
- Java Mail Sessions
- JMS Destinations

JDBC Datasources

A JDBC Datasource is a J2EE resource that you can create and manage using Sun ONE Application Server.

The JDBC API is the API for connectivity with relational database systems. The JDBC API has two parts:

- An application-level interface used by the application components to access databases.
- A service provider interface to attach a JDBC driver to the J2EE platform.

A JDBC `DataSource` object is the representation of a data source in the Java programming language. In basic terms, a data source is a facility for storing data. It can be as sophisticated as a complex database for a large corporation or as simple as a file with rows and columns. A JDBC Datasource is a J2EE resource that can be created and managed via Sun ONE Application Server.

For more information on JDBC Datasources, see "About JDBC Resources" on page 264.

Java Mail Sessions

JMS destinations are J2EE resources that can be created and managed via Sun ONE Application Server.

Many internet applications require the ability to send email notifications, so the J2EE platform includes the JavaMail API along with a JavaMail service provider that allows an application component to send internet mail. The JavaMail API has two parts:

- An application-level interface used by the application components to send mail
- A service provider interface used at the J2EE SPI level.

Java Mail Sessions are J2EE resources that can be created and managed via Sun ONE Application Server. For more information on Java Mail Sessions, see "About Java Mail Resources" on page 286.

JMS Destinations

Java Messaging Service (JMS) is a standard API for messaging that supports reliable point-to-point messaging as well as the publish-subscribe model. This specification requires a JMS provider that implements both point-to-point messaging and publish-subscribe messaging.

JMS provides for two general types of administered objects: connection factories and destinations. While both encapsulate provider-specific information, they have very different uses within a JMS client. A connection factory is used to create connections to a message server, and destination objects are used to identify physical destinations used by the JMS messaging service.

About Java Naming and Directory Interface (JNDI)

This section discusses the Java Naming and Directory Interface (JNDI). Java Naming and Directory Interface (JNDI) is an application programming interface (API) for accessing different kinds of naming and directory services. J2EE components locate objects by invoking the JNDI lookup method.

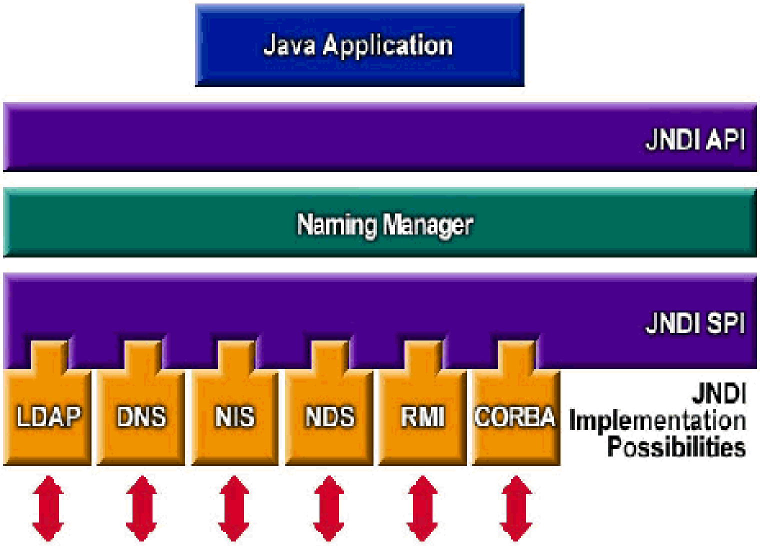
This section covers the following topics:

- JNDI Architecture
- J2EE Naming Services
- Naming References and Binding Information
- Naming References in J2EE Standard Deployment Descriptor
- JNDI Connection Factories

JNDI Architecture

The JNDI architecture consists of an Application Programmer's Interface (API) and a Service Provider Interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services. The following figure, "Overview of the JNDI Architecture," illustrates the services than can be accessed through the JNDI API:

Figure 10-1 Overview of the JNDI Architecture



J2EE Naming Services

A JNDI name is a people-friendly name for an object. These names are bound to their objects by the naming and directory service that is provided by a J2EE server. Because J2EE components access this service through the JNDI API, we usually refer to an object's people-friendly name as its JNDI name. The JNDI name of the Pointbase database is `jdbc/Pointbase`. When it starts up, Sun ONE Application Server reads information from configuration file and automatically adds JNDI database names to the name space.

J2EE application clients, enterprise beans, and web components are required to have access to a JNDI naming environment.

The application component's naming environment is a mechanism that allows customization of the application component's business logic during deployment or assembly. Use of the application component's environment allows the application component to be customized without the need to access or change the application component's source code.

A J2EE container implements the application component's environment, and provides it to the application component instance as a JNDI naming context. The application component's environment is used as follows:

- The application component's business methods access the environment using the JNDI interfaces. The application component provider declares in the deployment descriptor all the environment entries that the application component expects to be provided in its environment at runtime.
- The container provides an implementation of the JNDI naming context that stores the application component environment. The container also provides the tools that allow the deployer to create and manage the environment of each application component.
- A deployer uses the tools provided by the container to initialize the environment entries that are declared in the application component's deployment descriptor. The deployer can set and modify the values of the environment entries.
- The container makes the environment naming context available to the application component instances at runtime. The application component's instances use the JNDI interfaces to obtain the values of the environment entries.

Each application component defines its own set of environment entries. All instances of an application component within the same container share the same environment entries. Application component instances are not allowed to modify the environment at runtime. For more information on how J2EE containers such as the Web Container and EJB Containers use the JNDI naming service to look up objects, please see “Configuring J2EE Containers” on page 201.

Naming References and Binding Information

A resource reference is an element in a deployment descriptor that identifies the component’s coded name for the resource. More specifically, the coded name references a connection factory for the resource. In the example given in the following section, the resource reference name is `jdbc/SavingsAccountDB`.

The JNDI name of a resource and the name of the resource reference are not the same. This approach to naming requires that you map the two names before deployment, but it also decouples components from resources. Because of this de-coupling, if at a later time the component needs to access a different resource, you don’t have to change the name in the code. This flexibility also makes it easier for you to assemble J2EE applications from preexisting components.

The following table, “JNDI Lookups and Their Associated References,” lists JNDI lookups and their associated references for the J2EE resources used by Sun ONE Application Server.

Table 10-1 JNDI Lookups and Their Associated References

JNDI Lookup Name	Associated Reference
`java:comp/env`	Application environment entries
`java:comp/env/jdbc`	JDBC DataSource resource manager connection factories
`java:comp/env/ejb`	EJB References
`java:comp/UserTransaction`	UserTransaction references
`java:comp/env/mail`	JavaMail Session Connection Factories
`java:comp/env/url`	URL Connection Factories
`java:comp/env/jms`	JMS Connection Factories and Destinations
`java:comp/ORB`	ORB instance shared across application components

Naming References in J2EE Standard Deployment Descriptor

A naming reference is a string used by the application to look up an object in the given naming context. For each J2EE application, there is a naming context and the references are configured in the standard component deployment descriptors.This section describes the standard deployment descriptor features used in Sun ONE Application Server. This section covers the following topics:

- Application Environment Entries
- EJB References
- References to Resource Manager Connection Factories
- Resource Environment References
- UserTransaction References
- COSNaming Service

Application Environment Entries

Environment entries, defined using <env-entry>, provide a way of specifying deployment time parameters to J2EE applications. Note that in case of web applications, the servlet context initialization parameters could be defined using <context-param>, but <env-entry> is the preferred way because application deployers to configure such applications parameters by explicitly specifying the name, type and values for them.

The following sample describes the syntax of <env-entry> as specified in the J2EE standard deployment descriptors:

```
<env-entry>
<description> Send pincode by mail </description>
<env-entry-name> mailPincode </env-entry-name>
<env-entry-value> false </env-entry-value>
<env-entry-type> java.lang.Boolean </env-entry-type>
</env-entry>
```

The <env-entry-type> tag specifies a fully qualified class name for the entry. Here is a code snippet to lookup the <env-entry> using JNDI from an application component (a term referring to a servlet/JSP or an entity bean or an IIOP application client):

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

```
// one could use relative names into the sub-context
Context envContext = initContext.lookup("java:comp/env");
Boolean mailPincode = (Boolean)
envContext.lookup("mailPincode");
```

EJB References

Apart from deployment descriptor support, the JNDI naming service enables applications to use “logical” names (called EJB references) to map to the home interfaces of enterprise beans, as described in the following example:

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<ejb-ref-type> Entity </ejb-ref-type>
<home> com.wombat.empl.EmployeeRecordHome </home>
<remote> com.wombat.empl.EmployeeRecord </remote>
<ejb-link> EmployeeEJB </ejb-link>
</ejb-ref>
```

An application component such as a JSP could access the EJB home object using JNDI, as described in the following example:

```
Context initContext = new InitialContext();
Context envContext = initContext.lookup("java:comp/env");
Object result = envContext.lookup("ejb/EmplRecord");
EmployeeRecordHome emplRecordHome = (EmployeeRecordHome)
javax.rmi.PortableRemoteObject.narrow(result,
EmployeeRecordHome.class);
```

The ejb-ref-name element defines the string used in the application code (as in the above given example). The ejb-link element links this reference to target enterprise bean defined using the ejb-name element of the entity bean defined in the ejb-jar.xml. It is also possible to provide the linkage without modifying either the application deployment descriptor or the enterprise bean descriptor.

References to Resource Manager Connection Factories

A factory is an object that creates other objects on demand. A resource factory creates resource objects, such as database connections or message service connections. They are configured using <resource-ref> element in the standard deployment descriptors.

The following examples describe the use of factories:

Example A:

Declaration of a reference to a JDBC connection factory that returns objects of type javax.sql.DataSource:

```
<resource-ref>
<description> Primary database </description>
<res-ref-name> jdbc/primaryDB </res-ref-name>
<res-type> javax.sql.DataSource </res-type>
<res-auth> Container </res-auth>
</resource-ref>
```

Example B:

Here is an example reference to JavaMail Session resource factory:

```
<resource-ref>
<description> mail Session </description>
<res-ref-name> mail/Session </res-ref-name>
<res-type> javax.mail.Session </res-type>
<res-auth> Container </res-auth>
</resource-ref>
```

<res-type> is a fully-qualified class name of the resource factory. The <res-auth> variable can be assigned either Container or Application as a value. To know more about configuring Java Mail session resource factories, please see "About Java Mail Resources" on page 286.

If Container is specified, the web container handles the authentication before binding the resource factory to JNDI lookup registry. If Application is specified, the servlet must handle authentication programmatically. Different resource factories are looked up under a separate sub-context that describes the resource type, follows:

- `jdbc/` for a JDBC `javax.sql.DataSource` factory
- `jms/` for a JMS `javax.jms.QueueConnectionFactory` or `javax.jms.TopicConnectionFactory`
- `mail/` for a `JavaMail javax.mail.Session factory`
- `url/` for a `java.net.URL factory`

Here is a code snippet to get JDBC connection from an application component with the container handling the authentication:

```
InitialContext initContext = new InitialContext();
DataSource source =
(DataSource) initContext.lookup("java:comp/env/jdbc/primaryDB");
Connection conn = source.getConnection();
```

Please note that in order to ensure that for these resource references work, the `res-ref-name` must map to valid resource factory at runtime.

Resource Environment References

Resource environment references provide a way of accessing, via JNDI lookups, administered objects associated with a resource. For example, an application may need to access a JMS Destination object. The `<resource-env-ref>` element, defined in the standard deployment descriptors lets applications declare the resource requirements.

The main difference between `<resource-env-ref>` and `<resource-ref>` element is the absence of specific resource authentication requirement; both these elements have to be backed up by a resource factory descriptor.

Example:

```
<resource-env-ref>
<description> My Topic </description>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<res-env-ref-type> javax.jms.Topic </res-type>
</resource-env-ref>
```

The following piece of code allows you to access a JMS Topic object:

```
InitialContext initContext = new InitialContext();
javax.jms.Topic myTopic =
(javax.jms.Topic) initContext.lookup("java:comp/env/jms/MyTopic");
```

Note that in order these `resource-env-ref` variables to work, the administrators will have to make target resource factories available at run-time. For more information about accessing JMS Topics and Queue destinations, see Chapter 11, "Using the JMS Service."

UserTransaction References

J2EE requires that containers provide a `UserTransaction` object implementation at the JNDI name `java:comp/UserTransaction`. A `UserTransaction` object lets applications to start, commit and abort transactions.

To programmatically initiate and perform transactions, components obtain reference to the container's default transaction co-ordinator, by doing a JNDI lookup for `java:comp/UserTransaction`. The returned object implements `javax.transaction.UserTransaction` interface and can be used in the program to begin, commit, rollback and query status of transactions. JNDI implementation in Sun ONE Application Server supports such lookup of the transaction co-ordinator. For more information about the `javax.transaction.UserTransaction` interface, see "Using Transaction Services" on page 221.

Initial Naming Context

The naming support in Sun ONE Application Server is based primarily on J2EE 1.3, with a few added enhancements.When an application component creates the initial context, via `InitialContext()`, Sun ONE Application Server returns an object that serves as a handle to the application's naming environment. This object in turn provides sub-contexts for the `java:comp/env namespace`. Each application gets its own namespace, that is, `java:comp/env name` space is per application and objects bound in one application's namespace don't collide with objects bound in other applications.

COSNaming Service

The EJB interoperability protocol requires the use of the COSNaming protocol for lookup of EJB objects using the JNDI API.

EJB containers are required to be able to publish `EJBHome` object references in a CORBA `CosNaming` service. The `CosNaming` service must implement the IDL interfaces in the `CosNaming` module defined, and allow clients to invoke the resolve and list operations over IIOP.

The CosNaming service must follow the requirements in the CORBA Interoperable Name Service specification for providing the host, port and object key for its root NamingContext object. The CosNaming service must be able to service IIOP invocations on the root NamingContext at the advertised host, port and object key.

Client containers (that is, EJB, web or application client containers) are required to include a JNDI CosNaming service provider that uses the mechanisms defined in the Interoperable Name Service specification to contact the server's CosNaming service and to resolve the EJBHome object using standard CosNaming APIs. The JNDI CosNaming service provider may or may not use the JNDI SPI architecture. The JNDI CosNaming service provider must access the root NamingContext of the server's CosNaming service by creating an object reference from the following URL:

corbaloc:iiop:1.2@<host>:<port>/<objectkey> (where <host>, <port>, and <objectkey> are the values corresponding to the root NamingContext advertised by the server's CosNaming Service), or by using an equivalent mechanism.

At deployment time, the developer of the client container should obtain the host, port and object key of the server's CosNaming service and the CosNaming name of the server EJBHome object (for example, by browsing the server's namespace) for each ejb-ref element in the client components's deployment descriptor. The ejb-ref-name (which is used by the client code in the JNDI lokup call) should then be linked to the EJBHome object's CosNaming name. At runtime, the client component's JNDI lookup call uses the CosNaming service provider, which contacts the server's CosNaming service, resolves the CosNaming name, and returns the EJBHome object reference to the client component.

Since the EJBHome object’s name is scoped within the namespace of the CosNaming service that is accessible at the provided host and port, it is not necessary to federate the namespaces of the client and server containers.

The advantage of using CosNaming is better integration with the IIOP infrastructure that is already required for inter operability, as well as inter operability with non-J2EE CORBA clients and servers. Since CosNaming stores only CORBA objects it is likely that vendors will use other enterprise directory services for storing other resources.

Sun ONE Application Server incorporates all the naming resources of JNDI, based on the J2EE 1.3 specification.

CosNaming provider To support a global JNDI namespace (accessible to IIOP application clients), Sun ONE Application Server includes J2EE based CosNaming provider which supports binding of CORBA references (remote EJB references). The `InitialContext` returned to the IIOP clients is a CosNaming provider. An instance of Sun ONE Application Server server registers the entity beans for the IIOP clients to lookup and bind to.

Note that Sun ONE Application Server treats objects stored in CosNaming and the local JNDI naming environment are transient: that is, on each server startup as well as application reloading, all relevant objects are rebound to the namespace again. To know more about configuring support for CORBA / IIOP clients, see "Configuring the Server For CORBA / IIOP Clients" on page 329.

JNDI Connection Factories

For J2EE web applications, the deployment descriptor in the `web.xml` file is the placeholder for defining references to application environment entries, resource manager (such as SQL Data Source) connection factories, or EJBs. Applications look up such references using the JNDI `InitialNamingContext` provided by the J2EE containers. This makes applications portable to different application server environments by just making changes to the deployment descriptor, that is, without accessing or modifying the application's source code. Likewise, J2EE requires the deployment descriptors for entity beans (`ejb-jar.xml`) as well as the IIOP application clients (`application-client.xml`) to be the primary vehicles for these JNDI naming references.

A connection factory is an object that produces connection objects that enable a J2EE component to access a resource. The connection factory for a database is a `javax.sql.DataSource` object, which creates a `java.sql.Connection` object.

In Sun ONE Application Server, you can configure the means of accessing the following resources and resource factories:

- JDBC connection factories
- JMS Connection factories based on MQ
- JavaMail Session connection factories
- JCA Connector factories
- Generic, custom user-written resource object factories.
- Support for external resource repositories such as LDAP

All Sun ONE Application Server resource factories are specified within the `<resources> </resources>` tags in `server.xml` and have a JNDI name specified using the `jndi-name` attribute. This attribute is used to register the factory in the server-wide namespace. Deployers can map user-specified, application-specific resource reference names (declared within `resource-ref` or `resource-env-ref` elements) to these server-wide resource factories using the `resource-ref-mapping` element. This enables deployment time decisions to be made with regards to which JDBC drivers (and other resource factories) to use for a given application.

A custom resource accesses a local JNDI repository and an external resource accesses an external JNDI repository. Both types of resources need user-specified factory class elements, JNDI name attributes, etc. In this section, we will discuss how to configure JNDI connection factory resources, for J2EE resources, and how to access these resources.

The following topics are covered in this section:

- To Create a Custom Resource
- To Create an External JNDI Resource
- Accessing External JNDI Repositories
- Mapping Application Resource References
- About URL Connection Factory Resources
- Mapping Application Resource Environment References
- Mapping EJB References

To Create a Custom Resource

The `custom-resource` element defined in `server.xml` provides a way of specifying a custom server-wide resource object factory. Such object factories implement the `javax.naming.spi.ObjectFactory` interface. This element associates a JNDI name (specified through the jndi-name sub-element like other Sun ONE Application Server resources) to be used in the server-wide namespace, its type, name of the resource factory class and a set of standard properties used to instantiate the same.

The following example illustrates the implementation of the `javax.naming.spi.ObjectFactory` interface:

```
<resources> <custom-resource jndi-name="test/myBean"
res-type="test.MyBean"factory-class="test.MyBeanFactory"
enabled="true">
```

```
<property name="foo" value="test custom bean prop" />
</custom-resource>
</resources>
```

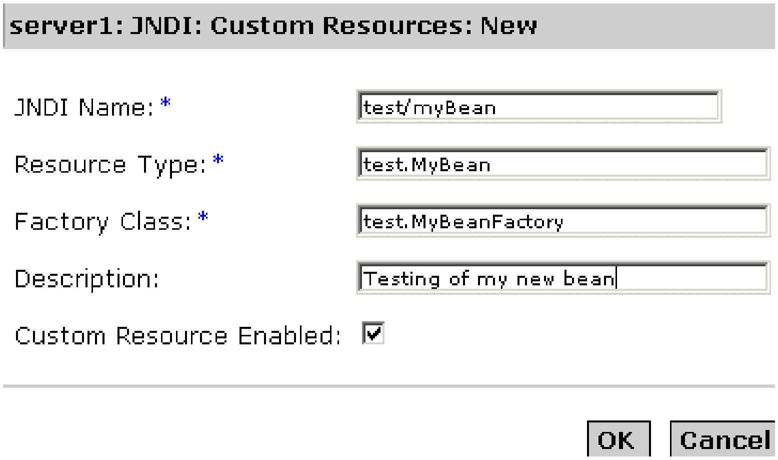
You need to ensure that the resource reference's environment references and EJB references are linked to the configured server-wide resources defined using the `custom-resource` and `external-jndi-resource` tags in `server.xml`. Dynamic redeployment of application components is an issue for the JNDI naming environment. Sun ONE Application Server will release all the application specific references and rebind all the new references into the newly installed application's naming context.

To create a custom resource using the Administration interface:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance whose JNDI configuration you want to modify.

2. Open the JNDI tab and click Custom Resources. If any custom resources have been created already, they will be listed in the right pane. To create a new custom resource, click New. You will see the "JNDI Custom Resources Page" in the right pane of the Administration interface:

Figure 10-2 JNDI Custom Resources Page



server1: JNDI: Custom Resources: New

JNDI Name: * test/myBean

Resource Type: * test.MyBean

Factory Class: * test.MyBeanFactory

Description: Testing of my new bean

Custom Resource Enabled: ☑

3. In the JNDI Name field, enter the name that is to be used to access the resource. This name will be registered in the JNDI naming service.

4. In the Resource Type field, enter a fully qualified type definition, as shown in the example above. Your Resource Type definition should follow this format: `xxx.xxx`.

5. In the Factory Class field, enter a factory class name for the custom resource you are creating. The Factory Class is the user-specified name for the factory class. This class implements the `javax.naming.spi.ObjectFactory` interface.

6. In the Description field, enter a description for the resource you're creating. This description is a string value and can comprise a maximum of 250 characters.

7. Mark the Custom Resource Enabled checkbox, to enable the custom resource.

8. Click OK to save your custom resource.

To Create an External JNDI Resource

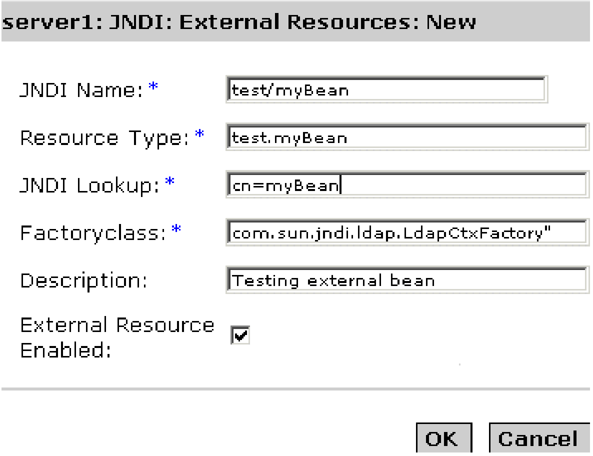
To create an external resource using the Administration interface:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance whose JNDI configuration you want to modify.

2. Open JNDI and select External Resources. If any external resources have been created already, they will be listed in the right pane. To create a new external resource, click New.

   You will see the following window, shown in the "JNDI External Resources Page" in the right pane of the Administration interface:

Figure 10-3 JNDI External Resources Page



server1: JNDI: External Resources: New

JNDI Name: *	test/myBean
Resource Type: *	test.myBean
JNDI Lookup: *	cn=myBean
Factoryclass: *	com.sun.jndi.ldap.LdapCtxFactory"
Description:	Testing external bean
External Resource Enabled:	☑

3. In the JNDI Name field, enter the name that is to be used to access the resource. This name will be registered in the JNDI naming service.

4. In the Resource Type field, enter a fully qualified type definition, as shown in the example above. Your Resource Type definition should follow this format: `xxx.xxx`.

5. In the JNDI Lookup field, enter the JNDI value to look up in the external repository. For example, if you are creating an external resource to connect to an external repository, to test a bean class, your JNDI Lookup could read `cn=`*testmybean*.

6. In the Factory Class field, enter a JNDI factory class external repository, for example, `com.sun.jndi.ldap`. This class implements the javax.naming.spi.`ObjectFactory` interface.

7. In the Description field, enter a description for the resource you’re creating. This description is a string value and can comprise a maximum of 250 characters.

8. Mark the External Resource Enabled checkbox, to enable the external resource.

9. Click OK to save your custom resource.

Accessing External JNDI Repositories

Often applications running on Sun ONE Application Server require access to resources stored in an external JNDI repository. For example, generic Java objects could be stored in an LDAP server as per the Java schema. External JNDI resource elements let users configure such external resource repositories. The external JNDI factory must implement `javax.naming.spi.InitialContextFactory` interface.

Example:

```
<resources>
<!-- external-jndi-resource element specifies how to access J2EE resources
-- stored in an external JNDI repository. The following example
-- illustrates how to access a java object stored in LDAP.
-- factory-class element specifies the JNDI InitialContext factory that
-- needs to be used to access the resource factory. property element
-- corresponds to the environment applicable to the external JNDI context
-- and jndi-lookup-name refers to the JNDI name to lookup to fetch the
-- designated (in this case the java) object.
-->
<external-jndi-resource jndi-name="test/myBean"
jndi-lookup-name="cn=myBean"
res-type="test.myBean"
factory-class="com.sun.jndi.ldap.LdapCtxFactory">
```

```
<property name="PROVIDER-URL" value="ldap://ldapserver:389/o=myObjects" />
<property name="SECURITY_AUTHENTICATION" value="simple" />
<property name="SECURITY_PRINCIPAL", value="cn=joeSmith, o=Engineering" />
<property name="SECURITY_CREDENTIALS" value="changeit" />
</external-jndi-resource>
</resources>
```

Mapping Application Resource References

Application-specific resource references must be mapped to pre-defined server-wide resource factories. The Sun ONE Application Server specific resource reference mapping element is used for this.

In the following example, we look at a web application's deployment descriptor `web.xml` where a resource reference is specified to a JDBC DataSource.

```
<resource-ref>
<res-ref-name> jdbc/EstoreDataSource </res-ref-name>
<res-type> javax.sql.DataSource </res-type>
<res-auth> Container </res-auth>
</resource-ref>
```

The desired `res-ref-name` can also be mapped to the container-wide Orcale JDBC connection resource factory, as follows:

```
<resource-ref>
<res-ref-name> jdbc/EstoreDataSource </resource-ref-name>
<jndi-name> jdbc/estore/InventoryDB </jndi-name>
</resource-ref>
```

About URL Connection Factory Resources

URL connection factories do not require any resource to be defined in server.xml. The jndi-name element of the corresponding Sun ONE Application Server application (web or ejb) deployment descriptor specifies the target URL.

For example, let us assume that a web application's deployment descriptor `web.xml` specifies a `java.net.URL` resource reference and this is mapped to the URL `http://www.`*sun.com/index.html* in `sun-web.xml`:

The mapping would be as follows:

```
<resource-ref>
<res-ref-name>myURL</res-ref-name>
<res-type>java.net.URL</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

```
<sun-web-app>
<resource-ref>
<res-ref-name>myURL</res-ref-name>
<jndi-name> http://www.sun.com/index.html </jndi-name>
</resource-ref>
</sun-web-app>
```

Mapping Application Resource Environment References

Application-specific resource environment reference declarations must be mapped to target resource objects available in the application server's runtime environment. The resource environment mapping element defined in the Sun ONE Application Server-specific configuration file lets deployers map as follows:

Example:

```
<resource-env-ref>
<description> My Topic </description>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<res-env-ref-type> javax.jms.Topic </res-type>
</resource-env-ref>
```

This reference is mapped to the `jms/iMQ/Topics/Stocks/SUNW` topic defined in `server.xml`. See the *Sun ONE Application Server Administrator’s Configuration File Reference* for more information.

```
<resource-env-ref-mapping>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<jndi-name> jms/iMQ/Topics/Stocks/SUNW </jndi-name>
</resource-env-ref-mapping>
```

Mapping EJB References

It's also possible to decouple the actual ejb-name used in the application code from the `ejb-name` used for the target enterprise bean. This is particularly useful when you do not want to modify the web application deployment descriptor, `web.xml` and use the `ejb-name` of the enterprise bean deployment descriptor. The Sun ONE Application Server specific configuration allows you to map the `ejb-ref-name` element to the target bean's `ejb-name` without using the `ejb-ref-mapping` element in the Sun ONE Application Server specific deployment descriptor.

Example:

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<ejb-ref-type> Entity </ejb-ref-type>
<home> com.wombat.empl.EmployeeRecordHome </home>
<remote> com.wombat.empl.EmployeeRecord </remote>
</ejb-ref>
```

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<jndi-name> AccountEJB </jndi-name>
</ejb-ref-mapping>
```

About Persistence Manager Resources

This module describes persistence and establishes a framework for the use of the pluggable Persistence Managers that are supported by Sun ONE Application Server.

This module covers the following topics:

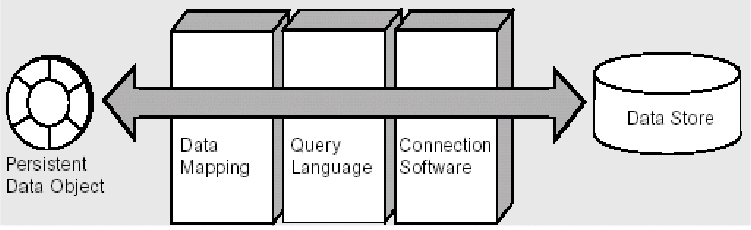
- What is Persistence?
- The Role of the Persistence Manager
- Pre-Deployment Bean Configuration
- Creating a New Persistence Manager

What is Persistence?

A key aspect of most business applications is the programmatic manipulation of persistent data; long-lived data stored outside of an application. Although persistent data is read into transient memory for the purpose of using or modifying it, it is written out to a relational database or flat file system for long-term storage.

In object-oriented programming systems, persistent data is represented in memory as one or more data objects manipulated by application code. In general, the correspondence between persistent data in a data store and its representation as a persistent data object in memory is achieved through a number of software layers as shown in the following figure, “Basic Persistence Scheme”:

Figure 10-4 Basic Persistence Scheme



Each data store has an interface to the outside world through driver software used to set up and maintain a connection between the data store and an application. With this connection established, a query language is used to retrieve information from the data store and read it into an application, or conversely, to write data from the application into the data store. Another layer provides a mapping between data objects in memory and the information in the data store.

Through this general scheme, programmers can represent persistent data as runtime objects to be used and manipulated by an application. The scheme supports all basic persistence operations—often abbreviated as CRUD:

- C - Creating persistent data (inserting in a data store)
- R - Retrieving persistent data (selecting from a data store)
- U - Updating persistent data
- D - Deleting persistent data

The Role of the Persistence Manager

The Persistence Manager (PM) is responsible for the persistence of entity beans with Container-managed Persistence, in the EJB container. The entity bean provider is responsible for providing the entity bean class as an abstract class. The Persistence Manager provider's tools are responsible for providing the concrete implementations. They can achieve this by sub-classing the abstract entity bean and related classes and providing an concrete implementation or by utilizing encapsulation and delegation.

The classes provided by the persistence manager's tools are responsible for managing the relationships between the entity beans and for managing the access to their persistent state. The PM tools are also responsible for providing the implementations of the `java.util.Collection` classes that are used in the maintaining the container managed relationships (CMRs).

Pre-Deployment Bean Configuration

The Enterprise Java Beans standard provides two types of persistence for Entity Beans. These are the Container Managed Persistence (CMP) and Bean Managed Persistence (BMP) The EJB 2.0 specification does not define a standard API between a EJB server and the persistence manager.

This section describes the integration requirements at deployment and code-generation. Deployment can be used to mean multiple things. Generally speaking, the deployment process can be thought of three distinct steps: Configuration, Code-generation and installation.

A number of properties must be specified for a bean, including the persistence mechanism used, persistence vendor, and the version in use, and additional information required by the persistence mechanism. Most of the persistence vendors have a concept of a project, which represents all the related beans and their dependant classes, which can be deployed as a single unit. There can be a vendor specific xml file per project.

The three standard files supported for deployment purposes include `ejb-jar.xml`, `sun-ejb-jar.xml` and `sun-cmp-mappings.xml`. Each EJB module with CMP beans in the `sun-ejb-jar.xml` must have a <`pm-descriptors`> with at least one <`pm-descriptor`> element, which specifies five more attributes. These five attributes are `pm-identifier`, `pm-version`, `pm-config`, `pm-class-generator`, and `pm-mapping-factory`.

The Sun ONE Application Server-specific descriptor (as in `sunEjb_jar_2_0.DTD`) defines the persistence manager related tags. A sample CMP descriptor might look like the one below as defined in the Sun ONE Application Server DTD:

```
PM descriptors contain one or more pm descriptors, but only of them
must be in use at any given time

-->

<!ELEMENT pm-descriptors ( pm-descriptor+, pm-inuse)>

<!--

pm-descriptor describes the properties for the persistence manager
```

```
associated with entity bean

-->

<!ELEMENT pm-descriptor ( pm-identifier, pm-version, pm-config?,
pm-class-generator?,

pm-mapping-factory?)>

<!--

This element describes the vendor who provided the PM implementation, for
example this could be Sun ONE Application Server Transparent Persistence,
TopLink, Versant or CocoBase:

-->

<!ELEMENT pm-identifier (#PCDATA)>

<!--

pm-version further specifies which version of PM vendor product to
be used

-->

<!ELEMENT pm-version (#PCDATA)>

<!--

pm-config specifies the vendor specific config file to be used

-->

<!ELEMENT pm-config (#PCDATA)>

<!--

pm-class-generator specifies the vendor specific concrete class
generator

This is the name of the class specific to a vendor:

-->

<!ELEMENT pm-class-generator (#PCDATA)>

<!--

pm-mapping-factory specifies the vendor specific mapping factory

This is the name of the class specific to a vendor:

-->

<!ELEMENT pm-mapping-factory (#PCDATA)>
```

Creating a New Persistence Manager

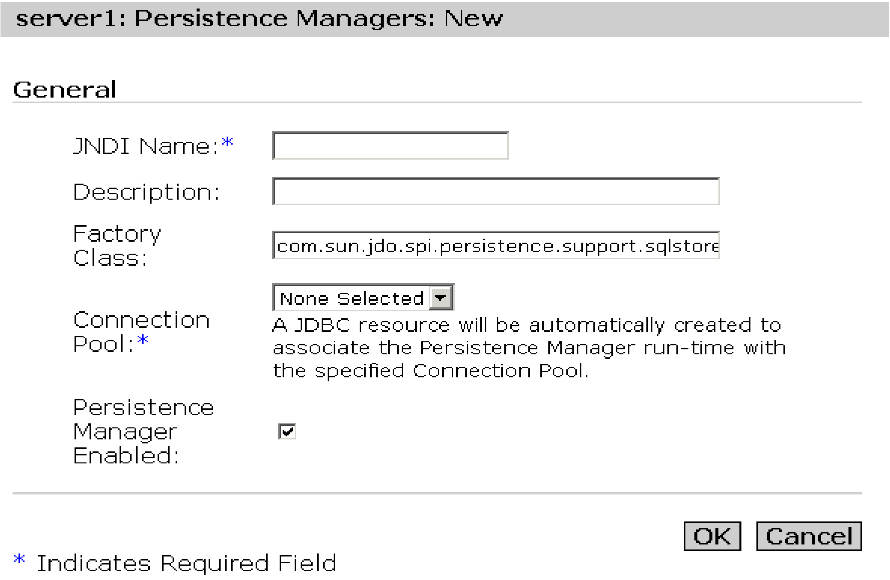
Using the Administration interface, you can create a new Persistence Manager instance. To create a new persistence manager instance:

1. From the left pane of the Administration interface, open the Sun ONE Application Server instance for which you want to create a new Persistence Manager. Click Persistence Manager from the list of server components displayed.

   If any persistence managers have been created for that specific instance of Sun ONE Application Server, you will see the list displayed in the right pane of the Administration interface.

2. To create a new Persistence Manager, click New. You will see the following window shown in the figure "Creating a New Persistence Manager":

Figure 10-5 Creating a New Persistence Manager



server1: Persistence Managers: New

General

JNDI Name:*

Description:

Factory Class: com.sun.jdo.spi.persistence.support.sqlstore

Connection Pool:* None Selected
A JDBC resource will be automatically created to associate the Persistence Manager run-time with the specified Connection Pool.

Persistence Manager Enabled: ☑

* Indicates Required Field

3. This is the JNDI Name used by the application server run-time to locate a particular Persistence Manager on behalf of an application. The name must the same as that defined in the entity bean's cmp-resource element of the sun specific deployment descriptor.

4. In the Description field, provide a description for the new persistence manager. The value for this field is string, and can comprise up to 250 characters.

5. In the Factory Class field, provide a factory class connection for the persistence manager. The `setEntityContext` looks up this connection factory through the JNDI name lookup. The factory class name is the classname of the Persistence Manager Factory that creates Persistence Manager instances. By default this will be set to the Sun ONE Application Server's internal Persistence Manager Factory class; if you use an alternative implementation you must ensure that this class is available in the server classpath.

6. From the Connection Pool drop-down list, select a database connection pool, into which the new persistence manager will be pooled. With connection pooling, an entity bean can request a single connection and use it to execute concurrent statements for multiple client threads. Just like any other database access, the Persistence Manager will use connection pooling to improve performance and scalability. Choose an existing connection pool or "None Selected" if you haven't yet created the pool.

   Note: a JDBC Resource will be automatically created to allow the PM runtime to bind to the connection pool using JNDI - the JNDI name of the JDBC Resource will be the same as the PM JNDI Name with a prefix of "PM." Deleting a Persistence Manager will also delete the associated JDBC Resource.

7. To enable the persistence manager, mark the Persistence Manager Enabled checkbox. The persistence manager is now enabled for the connection factory specified.

8. Click OK to save your changes.

About JDBC Resources

This module talks about the JDBC API in general, and JDBC resources and their implementation and usage in Sun ONE Application Server, in specific.

This module consists of the following sections:

- About the JDBC API
- About Database Access Models

- About JDBC Datasources
- About JDBC Connections
- About JDBC Transactions

About the JDBC API

The JDBC API is a Java API for accessing virtually any kind of tabular data. (As a point of interest, JDBC is the trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for “Java Database Connectivity.”) The JDBC API consists of a set of classes and interfaces written in the Java programming language that provide a standard API for tool/database developers and makes it possible to write database applications using an all-Java API.

The JDBC API makes it easy to send SQL statements to relational database systems and supports all dialects of SQL. But the JDBC 3.0 API goes beyond SQL, also making it possible to interact with other kinds of data sources, such as files that are outside of a database.

The value of the JDBC API is that an application can access virtually any data source and run on any platform with a Java virtual machine. In other words, with the JDBC API, it isn't necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an IBM DB2 database, and so on. You can write a single program using the JDBC API, and the program will be able to send SQL or other statements to the appropriate data source. And, with an application written in the Java programming language, you do not have to worry about writing different applications to run on different platforms. The combination of the Java platform and the JDBC API lets a programmer write once, and run the code from anywhere.

What Does The JDBC API Do?

A JDBC technology-based driver (JDBC driver) makes it possible to do three things:

- Establish a connection with a data source
- Send queries and update statements to the data source
- Process the results

The following code fragment gives a simple example of these three steps:

```
Context ctx = new InitialContext();
```

```
DataSource ds = (DataSource)ctx.lookup("jdbc/AcmeDB");
Connection con = ds.getConnection("myLogin", "myPassword");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
          int x = rs.getInt("a");
          String s = rs.getString("b");
          float f = rs.getFloat("c");
   }
```

About Database Access Models

The JDBC API supports both two-tier and three-tier models for database access. Sun ONE Application Server incorporates the more popular, two-tier database access model.

This section covers the following topics:

- Two-Tier Database Access Model
- Three-Tier Database Access Model

Two-Tier Database Access Model

In the two-tier database access model a Java applet or application talks directly to the data source using a DBMS-proprietary protocol. This access model requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This configuration is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

Three-Tier Database Access Model

In the three-tier database access model, the Java applet or application sends commands to a “middle tier” of services, which then sends the commands to the data source. The client application communicates with the middle tier via HTTP, RM, CORBA, or other calls. The middle tier communicates with the data store through a DBMS-proprietary protocol. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

About JDBC Datasources

A DataSource object is the representation of a data source in the Java programming language. In basic terms, a data source is a facility for storing data. It can be as sophisticated as a complex database for a large corporation or as simple as a file with rows and columns. A data source can reside on a remote server, or it can be on a local desktop machine. Applications access a data source using a connection, and a DataSource object can be thought of as a factory for connections to the particular data source that the DataSource instance represents. The DataSource interface provides two methods for establishing a connection with a data source.

A DataSource object has properties that identify and describe the data source it represents. Also, a DataSource object works with a JNDI naming service and is created, deployed, and managed separately from the applications that use it. A driver vendor will provide a class that is a basic implementation of the DataSource interface as part of its JDBC 2.0 or 3.0 driver product.

This section covers the following topics:

- Properties Of a DataSource Object
- Registering a JDBC Resource

Properties Of a DataSource Object

A DataSource object has a set of properties that identify and describe the real world data source that it represents. These properties include information like the location of the database server, the name of the database, the network protocol to use to communicate with the server, and so on. DataSource properties follow the JavaBeans design pattern and are usually set when a DataSource object is deployed.

To encourage uniformity among DataSource implementations from different vendors, the JDBC 2.0 API specifies a standard set of properties and a standard name for each property.

An instance of a class that implements the DataSource interface represents one particular data source. Every connection produced by that instance will reference the same data source. In a basic DataSource implementation, a call to the method DataSource.getConnection returns a connection object that, like the connection object returned by the DriverManager facility, is a physical connection to the data source.

JNDI provides a uniform way for an application to find and access remote services over the network. The remote service may be any enterprise service, including a messaging service or an application-specific service, but, of course, a JDBC application is interested mainly in a database service. Once a DataSource object is created and registered with a JNDI naming service, an application can use the JNDI API to access that DataSource object, which can then be used to connect to the data source it represents.

DataSource objects that implement connection pooling likewise produce a connection to the particular data source that the DataSource class represents. The connection object that the method DataSource.getConnection returns, however, is a handle to a PooledConnection object rather than being a physical connection. An application uses the connection object just as it usually does and is generally unaware that it is in any way different. Connection pooling has no effect whatever on application code except that a pooled connection, as is true with all connections, should always be explicitly closed. When an application closes a connection that is pooled, the connection joins a pool of reusable connections. The next time DataSource.getConnection is called, a handle to one of these pooled connections will be returned if one is available. Because connection pooling avoids creating a new physical connection every time one is requested, it can help to make applications run significantly faster.

A DataSource class can likewise be implemented to work with a distributed transaction environment. An EJB server, for example, supports distributed transactions and requires a DataSource class that is implemented to interact with it. In this case, the DataSource.getConnection method returns a Connection object that can be used in a distributed transaction. As a rule, EJB servers provide support for connection pooling as well as distributed transactions. Like connection pooling, transaction management is handled internally, so using distributed transactions is easy. The only requirement is that when a transaction is distributed (this involves two or more data sources), the application cannot call the transaction methods as either commit or rollback. It also cannot put the connection in auto-commit mode. The reason for these restrictions is that a transaction manager begins and ends a distributed transaction under the covers, so an application cannot do anything that would affect the time that a transaction begins or ends. To know more about Java transactions, please see Chapter 9, “Using Transaction Services.”

Registering a JDBC Resource

You can register a JDBC resource with Sun ONE Application Server using either the Administration interface or the command-line interface.

This section covers the following topics:

- Registering a Resource Using the Command Line
- Registering a Resource Using the Administration Interface

Registering a Resource Using the Command Line

To register a JDBC resource using the command line interface, run the following command:

```
./asadmin create-jdbc-resource
```

The XML snippet, for registering a JDBC resource, would have to specify a few attributes, as below (excerpted from `sun-server_7_0.dtd`).

```
<!-- JDBC javax.sql.DataSource resource definition -->

<!ELEMENT jdbc-resource (description?, property*)>

<!ATTLIST jdbc-resource  jndi-name CDATA     #REQUIRED

pool-name CDATA     #REQUIRED

enabled %boolean; 'true'>
```

Note that all this specifies, is the symbolic name with which applications will refer to this data source, from inside J2EE applications. `pool-name` attribute points to a named pool definition, that specifies all aspects of database connectivity. The enabled attribute can be used by the administrator to turn off some resources.

Registering a Resource Using the Administration Interface

To register a datasource using the Administration interface:

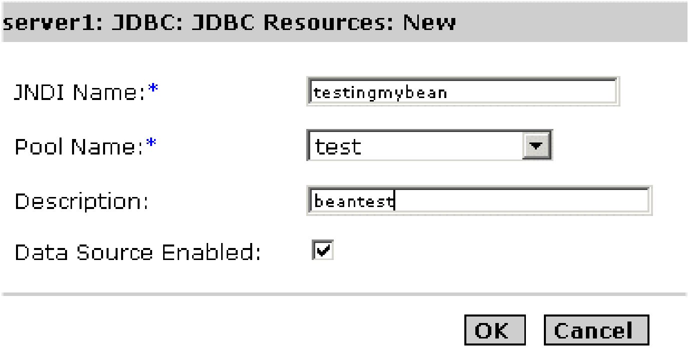
1. In the left pane of the Administration interface, open the Sun ONE Application Server instance for which you want to register a JDBC resource.

2. Open JDBC.

3. Under JDBC, click JDBC Resource.

4. In the right pane, click New. The page for creating a new JDBC resource, shown in the figure "Creating a New JDBC Resource" appears in the right pane.

Figure 10-6 Creating a New JDBC Resource



server1: JDBC: JDBC Resources: New

JNDI Name:* testingmybean

Pool Name:* test

Description: beantest

Data Source Enabled: ☑

5. Provide the JNDI name for the resource that you are creating.

   JDBC Resources are stored in the JNDI repository and are accessed using the JNDI name. The JNDI name has an explicit root at `Java:comp:env/` so you do not need to specify that part of the name. It is recommended that JDBC Resources (DataSources) are stored under the '`jdbc`' sub-context so your JNDI name would be similar to `jdbc/EmployeeDB_DS`.

6. Select a pool name for the new datasource, from the list of pool names in the Pool Name drop-down list. All registered connection pools will appear in this drop-down list. The pool name that you select points to a named pool definition, that specifies all aspects of database connectivity. More than one JDBC resource can use a single pool definition. To know more about how to configure JDBC connection pools, see "Creating a New JDBC Connection Pool Using the Administration Interface" on page 274.

7. In the Description field, provide a brief description that describes the purpose of the data source. Your description must not exceed 250 characters.

8. Mark the Enabled checkbox to enable or disable a datasource. A datasource cannot be used to connect to a database unless it is enabled.

9. Click OK to register the new datasource, or click Cancel to cancel the new datasource. When you click cancel, you are returned to the main JDBC resources page, from where you can create a new datasource again.

About JDBC Connections

A Connection object represents a connection with a database. A connection session includes the SQL statements that are executed and the results that are returned over that connection. A single application can have one or more connections with a single database, or it can have connections with many different databases.

A user can get information about a Connection object's database by invoking the `Connection.getMetaData` method. This method returns a `DatabaseMetaData` object that contains information about the database's tables, the SQL grammar it supports, its stored procedures, the capabilities of this connection, and so on.

An application uses a Connection object produced by a DataSource object. As is always the case, an application should include a “finally” block to assure that connections are closed even if an exception is thrown. This is even more important if the Connection object is a pooled connection because it makes sure that a valid connection will always be put back into the pool of available connections. The following code fragment, in which con is Connection object, is an example of a finally block that closes a connection if it is valid.

```
finally{

        if (con != null) con.close();

}
```

Note that a `finally` block appears after a try/catch block, as shown in the following example, where `ds` is a DataSource object.

```
try {

        Connection con = ds.getConnection("user", "secret");

        // . . . code to do the application's work

} catch {

        // . . . code to handle an SQLException

} finally {

        if (con != null) con.close();

}
```

This section covers the following topics:

- About JDBC URLs
- Configuring JDBC Connection Pools
- About Connection Pooling

- Monitoring JDBC Connection Pooling
- About Connection Sharing

About JDBC URLs

A URL (Uniform Resource Locator) gives information for locating a resource on the Internet. It can be thought of as an address.

A JDBC URL provides a way of identifying a data source so that the appropriate driver will recognize it and establish a connection with it. Driver writers are the ones who actually determine what the JDBC URL that identifies a particular driver will be. Users do not need to worry about how to form a JDBC URL; they simply use the URL supplied with the drivers they are using. JDBC's role is to recommend some conventions for driver writers to follow in structuring their JDBC URLs.

Since JDBC URLs are used with various kinds of drivers, the conventions are, of necessity, very flexible. First, they allow different drivers to use different schemes for naming databases. The ODBC sub-protocol, for example, lets the URL contain attribute values (but does not require them).

Second, JDBC URLs allow driver writers to encode all necessary connection information within them. This makes it possible, for example, for an applet that wants to talk to a given database to open the database connection without requiring the user to do any system administration chores.

Third, JDBC URLs allow a level of indirection. This means that the JDBC URL may refer to a logical host or database name that is dynamically translated to the actual name by a network naming system. This allows system administrators to avoid specifying particular hosts as part of the JDBC name. There are a number of different network name services, and there is no restriction about which ones can be used.

The standard syntax for JDBC URLs is shown here. It has three parts, which are separated by colons.

`jdbc:`<*subprotocol*>`:`<*subname*>

The three parts of a JDBC URL are broken down as follows:

- jdbc-the protocol:

  The protocol in a JDBC URL is always jdbc.

- `<subprotocol>`

The name of the driver or the name of a database connectivity mechanism, which may be supported by one or more drivers. A prominent example of a sub-protocol name is ODBC, which has been reserved for URLs that specify ODBC-style data source names. For example, to access a database through a JDBC-ODBC bridge, one might use a URL such as `jdbc:odbc:fred.`

In this example, the subprotocol is ODBC, and the subname fred is a local ODBC data source.

If one wants to use a network name service (so that the database name in the JDBC URL does not have to be its actual name), the naming service can be the subprotocol. So, for example, one might have a URL such as:

`jdbc:dcenaming:accounts-payable`

In this example, the URL specifies that the local DCE naming service should resolve the database name accounts-payable into a more specific name that can be used to connect to the real database.

- `<subname>`:

  A way to identify the data source. The subname can vary, depending on the subprotocol, and it can have any internal syntax the driver writer chooses, including a `sub-subname`. The point of a `subname` is to give enough information to locate the data source. In the previous example, fred is enough because ODBC provides the remainder of the information. A data source on a remote server requires more information, however. If the data source is to be accessed over the Internet, for example, the network address should be included in the JDBC URL as part of the `subname` and should adhere to the following standard URL naming convention:

  `//`*hostname*`:`*port*`/`*subsubname*

  Supposing that `dbnet` is a protocol for connecting to a host on the Internet, a JDBC URL might look like this:

  `jdbc:`*dbnet*`://`*wombat:356*`/`*fred*

Configuring JDBC Connection Pools

Sun ONE Application Server allows users to create named JDBC connection pools. A JDBC connection pool defines the properties used to create a connection pool. A pool definition is named, and a definition can be reused to configure multiple JDBC resources. Each named pool definition results in an instantiation of a physical pool at server start-up. If two or more JDBC resources point to the same pool definition, they will be using the same pool of connections at run time.

You can create and configure JDBC connection pools using the Administration interface and the Command Line Interface, as detailed in the following sections:

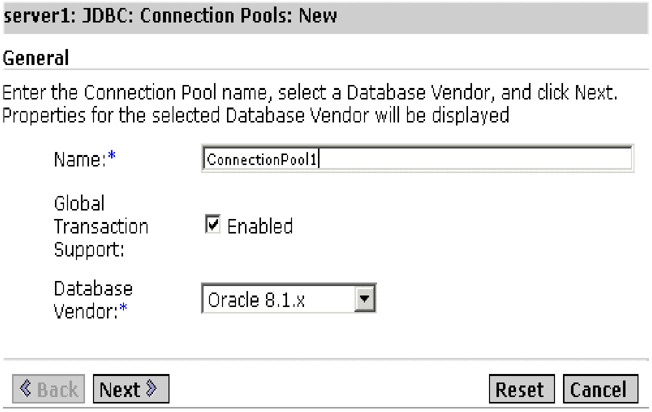
- Creating a New JDBC Connection Pool Using the Administration Interface
- Creating a New JDBC Connection Pool Using the Command-Line Interface
- Using the Command Line Interface to Manage JDBC Connection Pools

Creating a New JDBC Connection Pool Using the Administration Interface

To create a new JDBC connection pool using the Administration interface, perform the following tasks:

1. In the left pane of the Administration interface, open the Sun ONE Application Server instance for which you want to create a JDBC connection pool.
2. Select JDBC from the list of J2EE services listed under your Sun ONE Application Server, and open the `ConnectionPools` tab under it. You will see the figure "Creating a New JDBC Connection Pool" in the right pane of the Administration interface.

Figure 10-7 Creating a New JDBC Connection Pool



server1: JDBC: Connection Pools: New

General

Enter the Connection Pool name, select a Database Vendor, and click Next.
Properties for the selected Database Vendor will be displayed

Name:* ConnectionPool1

Global Transaction Support: ☑ Enabled

Database Vendor:* Oracle 8.1.x

3. In the Name field, provide a JNDI name of the connection pool that you are creating.
4. Mark the Global Transaction Support Enabled checkbox to enable global Transaction support for the new connection pool. Connection pools that are capable of participating in global transactions, and are referred to as XA-capable connection pools.

5. Select a database vendor, from the Database Vendor drop-down list, and click Next. You need to configure connection pool settings in the screen that is displayed next.

Configuring Connection Pool Settings

To configure pool settings, perform Step 1to Step 5 as given in "Creating a New JDBC Connection Pool Using the Administration Interface" on page 274. When you click Next, as described in Step 5, a new page appears in the right pane of the Administration interface. It contains the following sections:

- General
- Properties
- Pool Settings
- Connection Validation
- Transaction Isolation

In the General section of this page, specify the values for the parameters provided, based on the guidelines given in the following table:

Table 10-2 General Settings

Parameter	Description
Name	The name of the connection pool.
DataSource ClassName	The vendor-specific classname that implements the DataSource and / or XADataSource APIs.
Description	The description of the connection pool.

In the Properties section of this page you specify standard and proprietary JDBC connection pool properties; many of these properties are optional. By default the names of all of the standard properties are provided. You will need to consult your database vendor's documentation to determine which standard and vendor specific properties are required.

In the Pool Settings section of this window, specify the values for the parameters provided, based on the guidelines given in the following table:

Table 10-3 Connection Pool Settings

Parameter	Description
Steady Pool Size	Specify the minimum number of connections that must be maintained in the pool. When a connection is given to a requesting thread, it is removed from the pool, reducing the current pool size. The steady pool size also refers to the number of entries that will be added to the pool on server startup.
Max Pool Size	Specify the maximum number of connections that can be allowed in the pool at any given point in time.
Pool Resize Quantity	When the pool shrinks toward the steady pool size it is resized in batches. This value determines the size of the batch. Making this value too large will delay connection recycling, making it too small will be less efficient. Note, the pool capacity is only ever increased one connection at a time so this field does not effect increases in pool capacity.
Idle Timeout (secs)	The maximum time in seconds that a connection can remain idle in the pool. After this time, the pool implementation can close this connection.
Max Wait tim	The amount of time the caller will wait before getting a connection timeout. The default wait time is long, which means that a caller can wait for a long time.

In the Connection Validation and Transaction Isolation sections of this window, select the validation method and transaction isolation methods for the connection pool, based on the guidelines given in the following table:

Table 10-4 Connection Validation and Transaction Isolation

Parameter	Description
Connection Validation Required	If this field is checked then connections will be validated before they are passed to the application. This allows the application server to automatically re-establish database connections in the case of the database becoming unavailable due to network failure or database server crash. Validation of connections will incur additional overhead and slightly reduce performance.

Table 10-4 Connection Validation and Transaction Isolation

Parameter	Description
Validation Method	There are three methods that the application server can employ to validate database connections; you need to understand the capabilities of your database to determine the appropriate one. The three validation methods are: • `auto-commit, meta-data` - the con.getAutoCommit() and con.getMetaData() methods are commonly used to validate a connection, unfortunately many JDBC drivers cache the results of these calls so they do not always provide a reliable validation. You should check with your vendor to determine whether these calls are cached or not. • `table`: this method requires the app server to perform a query on a user specified table. The actual query is "select (count *) from <table-name>". The table must exist and be accessible, though it doesn't require any rows. You should not use an existing table that has a large number of rows or a table that is already frequently accessed.
Table Name	If you select the last validation option, `table`, from the Validation Method drop-down list, specify the table name here.
Fail All Connections	Check this box to fail all connections in the pool and re-establish them if a single connection is determined to have failed. If left unchecked, connections will be individually re-established only when they are used.
Transaction Isolation	Allows you to select the transaction isolation level for this connection. If left unspecified the pool operates with default isolation level provided by the JDBC Driver.
Guarantee Isolation Level	Only applicable if the isolation level has been specified. This ensures that any connection taken from the pool will have the same isolation level. For example if the isolation level for the connection was changed programatically (for example, `con.setTransactionIsolation`) when last used this mechanism will change it back to the specified isolation level.

Creating a New JDBC Connection Pool Using the Command-Line Interface

This section describes, through the use of examples, how to create JDBC connection pools using the Command Line Interface.

The following table lists all the options that you need to create connection pools, such as server name, password. Sample values have been used in the following table. It is recommended that you keep the parameters specific to your installation of Sun ONE Application Server ready, before running the commands explained in this section.

Table 10-5 Options required for creating JDBC Connection Pools using the Command Line Interface

Description of the required option	Sample Value
Application server admin user name	***admin***
Application server admin password	*adminadmin*
Application server admin port	*8888*
Application server machine name	*sas.sun.com*
Application server instance name	*server1*
Datasource classname for the connection pool	*oracle.jdbc.xa.client.OracleXADataSource* Note: Use the datasource classname of the database for which you are creating the connection pool. The database used in this example is Oracle.
Jdbc resource description Sample	*Jdbc Resource*
Connection pool description Sample	*Jdbc Connection Pool*
Jdbc resource name	*jdbc/SampleJdbcResource*
Connection pool name	*SampleJdbcConnectionPool*
Database user name	*oracle*
Database password	*oracle*
Jdbc connection URL	*jdbc:oracle:thin:@oracleserver.sun.com:1521:ORA*

The following example makes use of the variables listed in the "Options required for creating JDBC Connection Pools using the Command Line Interface" table.

Example 1:

This example creates a JDBC connection pool called `SampleJdbcConnectionPool.` A two-step process is used, in this example, to create the JDBC connection pool, as follows:

- Step 1 - Create a Connection Pool

- Step 2 - Apply Changes To the Instance

Step 1 - Create a Connection Pool

The following is the command line interface syntax for creating a JDBC connection pool:

```
asadmin create-jdbc-connection-pool --user admin_user [--password
admin_password] [--host localhost] [--port 4848] [--secure | -s]
[--instance instancename] --datasourceclassname classname [--restype
res_type] [--steadypoolsize 8] [--maxpoolsize 32] [--maxwait 60000]
[--poolresize 2] [--idletimeout 300] [--isolationlevel isolation_level]
[--isisolationguaranteed] [--isconnectvalidatereq=false]
[--validationmethod auto-commit] [--validationtable tablename]
[--failconnection=false] [--description text] [--property
(name=value)[:name=value]*] connectionpool_id
```

For example, the following command will create a connection pool called `SampleJdbcConnectionPool`.

```
asadmin create-jdbc-connection-pool --user admin --password
adminadmin --host sas.sun.com --port 8888 --instance server1
--restype javax.sql.XADataSource --datasourceclassname
oracle.jdbc.xa.client.OracleXADataSource --description "Sample Jdbc
Connection Pool" --property
User="oracle":Password="oracle":URL="jdbc\:oracle\:thin\:@oracleser
ver.sun.com\:1521\:ORA" SampleJdbcConnectionPool
```

NOTE If you want to enable “Global Transaction Support” for the new connection pool, set `--restype javax.sql.XADataSource`. In the URL property, replace the colon `(:) with (\:)`

Once the JDBC connection pool is created successfully, you see the following message:.

```
Created the JDBC connection pool resource with id =
SampleJdbcConnectionPool
```

Step 2 - Apply Changes To the Instance

Now that you have successfully created a JDBC connection pool, you need to apply the changes to the current instance of Sun ONE Application Server.

The following is the syntax for applying the changes to your instance of Sun ONE Application Server.

```
asadmin reconfig --user admin_user [--password admin_password] [--host
localhost] [--port adminport] [--secure | -s]
[--discardmanualchanges=false|--keepmanualchanges=false] instancename
```

For example, the following command applies the changes to *server1*, the instance of Sun ONE Application Server.

```
asadmin reconfig --user admin --password adminadmin --host sas.sun.com
--port 8888 server1
```

Once the changes are applied to your instance of Sun ONE Application Server, you will see the following message.

```
Successfully reconfigured
```

Using the Command Line Interface to Manage JDBC Connection Pools

You can use the command line interface to manage JDBC connection pools and their properties, as discussed in this section:

Listing Connection Pools. The following command lists all the connection pools created for *server1*, the instance of Sun ONE Application Server used in Step 2.

```
asadmin list-jdbc-connection-pools --user admin --password adminadmin
--host sas.sun.com --port 8888 server1
```

Changing a JDBC Connection Pool Property. You can change a JDBC connection pool's property, for example, the `maxPoolSize` property, as follows:

1. Run the following command to get the value specified for the JDBC connection pool attribute `maxPoolSize.`

   ```
   asadmin get -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSize
   ```

   When you run this command, you will see the following result:

   ```
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSiz
   e = 32
   ```

   Change the value of MaxPoolSize to 80, by running the following command:

   ```
   asadmin set -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSize="80"
   ```

   Once the values are specified as indicated, you will see the following message:

   ```
   Attribute maxPoolSize set to 80
   ```

2. Apply the changes to your instance of Sun ONE Application Server using the following command:

   ```
   asadmin reconfig --user admin --password adminadmin --host
   sas.sun.com --port 8888 server1
   ```

Changing the User Property. In the following piece of sample code, you can change the property “`User`” from `oracle` to `System`.

```
asadmin create-jdbc-connection-pool --user admin --password adminadmin
--host sas.sun.com --port 8888 --instance server1 --restype
javax.sql.XADataSource --datasourceclassname oracle.jdbc.xa.client.OracleXADataSource
--description "Sample Jdbc Connection Pool" --property
User="oracle":Password="oracle":URL="jdbc\:oracle\:thin\:@oracleserver.sun.com\:1521\
:ORA" SampleJdbcConnectionPool
```

1. Run the following command to change the User property.

   ```
   asadmin set -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.property.User="System"
   ```

   The name of the user is changed from *Oracle* to *System*.

2. After you change the user name, run the following command to apply your changes:

   ```
   asadmin reconfig --user admin --password adminadmin --host
   sas.sun.com --port 8888 server1
   ```

Creating a JDBC resource called SampleJdbcResource. You can create a JDBC resource, as detailed below. The following is the syntax for creating a JDBC resource:

```
asadmin create-jdbc-resource --user admin_user [--password
admin_password] [--host localhost] [--port 4848] [--secure | -s]
[--instance instancename] --connectionpoolid id [--enabled=true]
[--description text] [--property (name=value)[:name=value]*] jndiname
```

1. Run the following command to create a JDBC resource called `SampleJdbcResource`.

   asadmin create-jdbc-resource --user *admin* --password *adminadmin* --host *sas.sun.com* --port *8888* --instance *server1* --description "*Sample Jdbc Resource*" --connectionpoolid SampleJdbcConnectionPool jdbc/SampleJdbcResource

   When you run this command, the JDBC resource is created, and you will see the following message:.

   ```
   Created the external JDBC resource with jndiname =
   jdbc/SampleJdbcResource
   ```

2. Next, you need to run the following command to apply the changes to your instance of Sun ONE Application Server.

   asadmin reconfig --user *admin* --password *adminadmin* --host *sas.sun.com* --port *8888 server1*

3. Run the following command to list all the JDBC resources in the instance *server1*.

   asadmin list-jdbc-resources --user *admin* --password *adminadmin* --host *sas.sun.com* --port *8888 server1*

About Connection Pooling

Applications can obtain connections by first looking up the DataSource using JNDI. A sample code fragment to accomplish this, is shown below:

```
InitialContext ctx  = new InitialContext();

DataSource ds = (DataSource)
ctx.lookup("java:comp/env/jdbc/employee_ds");
```

After obtaining the `DataSource`, the application component can obtain connections in two ways depending on the value set on the <`res-auth`> element in the J2EE deployment descriptor. If the value of this element is Container, then the application can obtain a connection using `ds.getConnection()` method (that is, without specifying any sign-on information). Otherwise, the application has to give the sign-on information to obtain the connection from the resource manager like `ds.getConnecion`.(*userName, password*).

All requests to `getConnection()` will be served from a pool. The JDBC Connection pool will be created based on a set of parameters described in server.xml. When the pool is created, the pool contains the number of connections that are available initially. So the `ds.getConnection()` request can be satisfied with the connections currently available in the pool. The next request (if none of the previous connections have been returned to the pool) will find the pool empty and will result in the creation of incremented connections, subject to the limit of the maximum connections specified for the pool. The pool implementation will keep track of the number of connections that are created. If a `getConnection()` request finds the pool empty, or if the number of created connections is equal to the maximum connections in the pool, then the current request is blocked. This happens only if connection sharing is not possible, and continues until a connection is returned to the pool.

The connection pool will continue to work properly even if the database crashes and comes up, while the server is still running. This can be accomplished only if you enable connection validation, as described in "Configuring Connection Pool Settings" on page 275.

Depending on the value that you select from the Validation Type drop-down list, the following parameters are executed by the pool implementation program:

- If you choose `auto-commit` as your connection validation type, the system checks to see if the connection is valid by performing a `conn.getAutoCommit()` method. If the method does not throw an SQLException then the Connection is assumed to be valid. `auto-commit` is the default option for this parameter.
- If you chose `meta-data` as your connection validation type, then the `conn.getMetaData()` method will be performed, to check for metadata on the connection. and if the method didn't throw SQLException then the Connection is assumed to be valid.
- If you choose `table` as your connection validation type, then the query "`Select * From <table-name>`" will be executed. If this call doesn't throw an SQLException then the connection is assumed to be valid.

If you enable the fail-all-connections property, then if any connection in the pool is found to be invalid, all connections will be closed and re-established. Otherwise, invalidation and re-establishment is lazy and happens on use of individual connections.

The pool implementation also provides the ability to recycle all available connections in the pool. So if connections are idle beyond the idle period specified, they will be closed, bringing the pool size to the steady pool size. If the pool is extremely idle, it may result in the container having to re-establish stale connection and always keep the pool populated with a steady pool of usable connections. You must keep this in mind while determining whether to set the steady pool size relative to the maximum pool size.

Monitoring JDBC Connection Pooling

You may want to monitor the pool operation periodically, to determine if the pool size configuration is working effectively. The following table lists all the JDBC connection pooling parameters that can be monitored.

Note that the mechanics of enabling monitoring, and the attributes that can be monitored are likely to be improved in forthcoming releases.

Table 10-6 JDBC Connection Pool Parameters for Monitoring

Attribute Name	Data Type	Description
`total-threads-waiting`	Integer	Total threads waiting for JDBC connection.
`total-outbound-connections`	Integer	Total JDBC connection validation failures
`total-connections-timed-out`	Integer	Total connection requests timed out.

About Connection Sharing

When multiple connections acquired by a J2EE application use the same resource manager, the pool implementation will provide connection sharing within the same transaction scope. The term transaction scope, can be understood from the following example:

Bean_A starts a transaction (Tx1) and obtains a connection. Bean_A then calls a method in Bean_B in the same transaction (Tx1). Now if Bean_B acquires a connection from the same `DataSource` and with the same sign-on information, it is clear that the same connection can be shared as only Bean_A will complete the transaction. Also, note that connections will be shared only if the resource sharing scope is set to `Shareable` in the J2ee deployment descriptor. If connection sharing is not desired, then the resource sharing scope must be set to `Unshareable`, in the deployment descriptor. Sun ONE Application Server provides connection sharing, as it provides better performance.

About JDBC Transactions

A transaction consists of one or more statements that have been executed, completed, and then either committed or rolled back. When the method `commit` or `rollback` is called, the current transaction ends and another one begins.

Generally a new Connection object is in auto-commit mode by default, meaning that when a statement is completed, the method `commit` will be called on that statement automatically. In this case, since each statement is committed individually, a transaction consists of only one statement. If the auto-commit mode has been disabled, a transaction will not terminate until the method `commit` or `rollback` is called explicitly, so it will include all the statements that have been executed since the last invocation of either commit or rollback. In this second case, all the statements in the transaction are committed or rolled back as a group.

The method `commit` makes permanent any changes an SQL statement makes to a database, and it also releases any locks held by the transaction. The method `rollback` will discard those changes.

In the case of two updates in one transaction, you might at times not want one change to take effect in an update, unless the other update is also affected. This can be accomplished by disabling auto-commit and grouping both updates into one transaction. If both updates are successful, then the `commit` method is called, making the effects of both updates permanent; if one fails or both fail, then the `rollback` method is called, restoring the values that existed before the updates were executed. Most JDBC drivers support transactions.

Classes and interfaces in the `javax.sql` package make it possible for Connection objects to be part of a distributed transaction, that is, a transaction that involves connections to more than one DBMS server. In order to be able to participate in distributed transactions, a Connection object must be produced by a `DataSource` object that has been implemented to work with the middle tier server's distributed transaction infrastructure. Unlike Connection objects produced by the `DriverManager`, a Connection object produced by such a `DataSource` object will have its auto-commit mode disabled by default. A standard implementation of a `DataSource` object, on the other hand, will produce Connection objects that are exactly the same as those produced by the `DriverManager` class.

When a Connection object is part of a distributed transaction, a transaction manager determines when the methods commit or rollback are called on it. Thus, when a Connection object is participating in a distributed transaction, an application should not do anything that affects when a connection begins or ends, such as calling the methods `Connection.commit` or `Connection.rollback`, or turning on the connection's auto-commit mode. These would interfere with the transaction manager's handling of the distributed transaction.

About Java Mail Resources

The JavaMail API allows for access to email messages contained in message stores, and for the creation and sending of email messages using a message transport. Specific support is included for Internet standard MIME messages. Access to message stores and transports is through protocol providers supporting specific store and transport protocols. The JavaMail API specification does not require any specific protocol providers, but JavaMail includes an IMAP message store provider and an SMTP message transport provider.

The JavaMail API provides a set of abstract classes defining objects that comprise a mail system. The API defines classes like Message, Store and Transport. The API can be extended and can be sub-classed to provide new protocols and to add functionality when necessary. In addition, the API provides concrete sub-classes of the abstract classes. These sub-classes, including `MimeMessage` and `MimeBodyPart`, implement widely used Internet mail protocols.

The JavaMail API draws heavily from IMAP, MAPI, CMC, c-client and other e-mail messaging system APIs. The JavaMail API supports many different messaging system implementation like, different message stores, different message formats, and different message transports. The JavaMail API provides a set of base classes and interfaces that define the API for client applications. Developers can sub-class JavaMail classes to provide the implementations of particular messaging systems, such as IMAP, POP3, and SMTP.

The following topics are covered in this section:

- About the JavaMail Message-handling Process
- About the Architectural Components of JavaMail
- About JavaBeans Activation Framework (JAF)
- About JavaMail Configuration Parameters
- J2EE Deployment Descriptor for JavaMail Session References
- Entries in Sun ONE Application Server Deployment Descriptor
- Creating a New JavaMail Session
- Configuring Advanced Resource Properties

About the JavaMail Message-handling Process

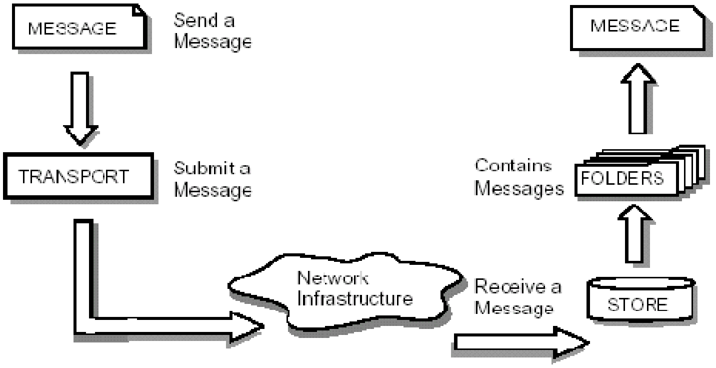
The JavaMail API performs the following functions, which comprise the standard mail handling process for a typical client application:

- Create a mail message consisting of a collection of header attributes and a block of data of some known data type as specified in the Content-Type header field. JavaMail uses the Part interface and the Message class to define a mail message. It uses the JAF-defined DataHandler object to contain data placed in the message.
- Create a Session object, which authenticates the user, and controls access to the message store and transport.
- Send the message to its recipient list.
- Retrieve a message from a message store.
- Execute a high-level command on a retrieved message. High-level commands like view and print are intended to be implemented via JAF-Aware JavaBeans.

NOTE Currently, the JavaMail framework does not define mechanisms that support message delivery, security, disconnected operation, directory services or filter functionality.

The following figure illustrates how the JavaMail API handles the message handling process:

Figure 10-8 Message Handling Process of the Java Mail API



The JavaMail API is configured by using a static factory method to create a `javax.mail.Session`. Sun ONE Application Server requests a Session object using JNDI, and lists its need for a Session object in its deployment descriptor using a `resource-ref` element. A JavaMail API Session object is considered a resource factory.

Message transport is provided that is capable of handling addresses of type `javax.mail.internet.InternetAddress` and messages of type `javax.mail.internet.MimeMessage`. The default message transport must be properly configured to send such messages using the send method of the `javax.mail.Transport` class.

The Abstract Layer of the JavaMail API declares classes, interfaces and abstract methods intended to support mail handling functions that all mail systems support. API elements comprising the Abstract Layer are intended to be sub-classed and extended as necessary in order to support standard data types, and to interface with message access and message transport protocols as necessary.

The internet implementation layer implements part of the abstract layer using internet standards - `RFC822` and `MIME`.

About the Architectural Components of JavaMail

This section describes the major components comprising the JavaMail architecture, given in the following topics:

- The Message Class
- Message Storage and Retrieval
- Message Composition and Transport

The Message Class

The Message class is an abstract class that defines a set of attributes and a content for a mail message. Attributes of the Message class specify addressing information and define the structure of the content, including the content type. The content is represented as a DataHandler object that wraps around the actual data.

The Message class implements the Part interface. The Part interface defines attributes that are required to define and format data content carried by a Message object, and to interface successfully to a mail system. The Message class adds From, To, Subject, Reply-To, and other attributes necessary for message routing via a message transport system. When contained in a folder, a Message object has a set of flags associated with it. JavaMail provides Message subclasses that support specific messaging implementations.

The content of a message is a collection of bytes, or a reference to a collection of bytes, encapsulated within a Message object. JavaMail has no knowledge of the data type or format of the message content. A Message object interacts with its content through an intermediate layer—the JavaBeans Activation Framework (JAF). This separation allows a Message object to handle any arbitrary content and to transmit it using any appropriate transmission protocol by using calls to the same API methods. The message recipient usually knows the content data type and format and knows how to handle that content.

The JavaMail API also supports multipart Message objects, where each Bodypart defines its own set of attributes and content.

Message Storage and Retrieval

Messages are stored in Folder objects. A Folder object can contain sub-folders as well as messages, thus providing a tree-like folder hierarchy. The Folder class declares methods that fetch, append, copy and delete messages. A Folder object can also send events to components registered as event listeners.

The Store Class

The Store class defines a database that holds a folder hierarchy together with its messages. The Store class also specifies the access protocol that accesses folders and retrieves messages stored in folders. The Store class also provides methods to establish a connection to the database, to fetch folders and to close a connection. Service providers implementing Message Access protocols (IMAP, POP3 etc.) start off by sub-classing the Store class. A user typically starts a session with the mail system by connecting to a particular Store implementation.

Message Composition and Transport

A client creates a new message by instantiating an appropriate Message subclass. It sets attributes like the recipient addresses and the subject, and inserts the content into the Message object. Finally, it sends the Message by invoking the Transport.send method. The Transport class models the transport agent that routes a message to its destination addresses. This class provides methods that send a message to a list of recipients. Invoking the Transport.send method with a Message object identifies the appropriate transport based on its destination addresses.

The Session Class

The Session class defines global and per-user mail-related properties that define the interface between a mail-enabled client and the network.

JavaMail system components use the Session object to set and get specific properties. The Session class also provides a default authenticated session object that desktop applications can share. The Session class is a final concrete class. It cannot be sub-classed. The Session class also acts as a factory for Store and Transport objects that implement specific access and transport protocols. By calling the appropriate factory method on a Session object, the client can obtain Store and Transport objects that support specific protocols.

About JavaBeans Activation Framework (JAF)

JavaMail uses the JavaBeans Activation Framework (JAF) in order to encapsulate message data, and to handle commands intended to interact with that data. Interaction with message data should take place via JAF-aware JavaBeans, which are not provided by the JavaMail API.

With the JavaBeans Activation Framework standard extension, developers who use Java technology can take advantage of standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and to instantiate the appropriate bean to perform said operation(s). For example, if a browser obtained a JPEG image, this framework would enable the browser to identify that stream of data as an JPEG image, and from that type, the browser could locate and instantiate an object that could manipulate, or view that image.

The JavaBeans Activation Framework API supports various MIME data types. The JavaMail API must include `javax.activation.DataContentHandlers` for the following MIME data types corresponding to the Java programming language type indicated in the following table.

Table 10-7 JavaMail API MIME Data Type to Java Type Mappings

MIME Type	Java Type
Text/Plain	java.lang.String
Multipart/	javax.mail.internet.MIME.Multipart
Message/rfc822	javax.mail.internet.MIME.Message

The JavaBeans Activation Framework integrates support for MIME data types into the Java platform. MIME byte streams can be converted to and from Java programming language objects, using avax.activation.DataContentHandlerobjects. JavaBeans components can be specified for operating on MIME data, such as viewing or editing the data. The JavaBeans Activation Framework also provides a mechanism to map filename extensions to MIME types. The JavaBeans Activation Framework is used by the JavaMail API to handle the data included in email message. Typical J2EE applications will not need to use the JavaBeans Activation Framework directly, although applications making sophisticated use of email may need it.

About JavaMail Configuration Parameters

The following configuration parameters are used by JavaMail Resources in Sun ONE Application Server. These configuration parameters are name, value pairs that would be read from the `mail-resource` element of the `server.xml` file.

- JNDI Name

  The JNDI Name specifies the name with which this mail resource will be referenced from a J2EE Application.

- Enabled

  The `enabled` configuration parameter specifies whether this mail resource will be published in the JNDI tree and can be referenced. If a J2EE application references a resource which is disabled, it will receive a `NameNotFoundException` exception.

- store-protocol

  Specifies the default Message Access Protocol. The `Session.getStore()` method returns a Store object that implements this protocol. The client can override this property and explicitly specify the protocol with the `Session.getStore(String protocol)` method.

- store-protocol class

  Specifies the class name that implements the store protocol specified above. The default for this class is `com.sun.mail.imap.IMAPStore`.

- transport-protocol

Specifies the default Transport Protocol. The `Session.getTransport()` method returns a Transport object that implements this protocol. The client can override this property and explicitly specify the protocol by using `Session.getTransport(String protocol)` method.

- transport-protocol class

  Specifies the class name that implements the transport protocol specified above. The default for this class is `com.sun.mail.smtp.SMTPTransport`.

- host

  Specifies the default Mail server. The Store and Transport object's connect methods use this property, if the protocol specific host property is absent, to locate the target host.

- user

  Specifies the username to provide when connecting to a Mail server. The Store and Transport object's connect methods use this property, if the protocol specific username property is absent, to obtain the username.

- from

  Specifies the return address of the current user. Used by the `InternetAddress.getLocalAddress` method to specify the current user's email address.

- debug

  Specifies the initial debug mode. Setting this property to true will turn on debug mode, while setting it to false turns debug mode off.

- mail-<protocol>-host

  Specifies the protocol-specific default Mail server. This overrides the `mail.host` property. This property could be set depending on the value of the store-protocol attribute. For the values of imap or POP for store-protocol, you need to add properties with the name `mail.imap.host` or `mail.pop3.host`, respectively. The value for the specific property should be set as it is appropriate in your mail system configuration. For example, with store-protocol set to IMAP, the property name: `mail-imap-host` would bear the value: `spaceduck.acme.com`.

- mail-<protocol>-user

Specifies the protocol-specific default user name for connecting to the Mail server. This overrides the `mail.user` property. So this property could be `mail.imap.user` or `mail.pop3.user` depending on the selection of store-protocol attribute. For example, with store-protocol set to IMAP, the property name: `mail-imap-user` would bear the value `fredbloggs`.

J2EE Deployment Descriptor for JavaMail Session References

Once a JavaMail Resource is registered with the server, it can be referenced in any J2EE application component using a JNDI lookup. In order to deploy an application that references resource manager connection factories, the component provider must declare all the resource manager connection factory references in the standard J2EE 1.3 deployment descriptor.

The complete J2EE1.3 descriptor elements for JavaMail reference are described below:

```
<resource-ref>
    <description>
          JavaMail resource used for sending my mail
     </description>
     <res-ref-name>mail/MyMailSession</res-ref-name>
     <res-type>javax.mail.Session</res-type>
     <res-auth>Container</res-auth>
     <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Entries in Sun ONE Application Server Deployment Descriptor

For each deployed component that refers to a mail resource, the deployer must map the resource name used in the component to the actual jndi name with which the DataSource is registered with the naming service. The deploy tool is expected to help the deployer to easily make this mapping. This mapping is registered in the Sun ONE Application Server specific xml. The fragment of the Sun ONE Application Server specific XML containing the mapping is shown below:

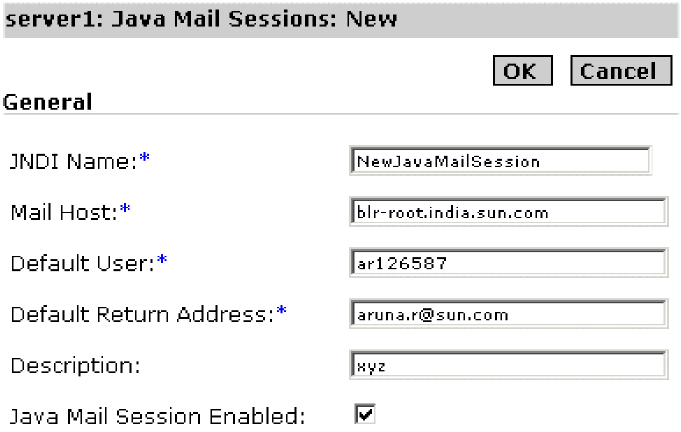
```
<resource-ref>
    <res-ref-name>mail/MyMailSession</res-ref-name>
    <jndi-name>mail/Session</jndi-name>
</resource-ref>
```

Creating a New JavaMail Session

You can configure JavaMail sessions using the Administration interface. To create and configure a new JavaMail session, perform the following tasks:

1. In left pane of the Administration interface, expand the Sun ONE Application Server instance, for which you want to create a new JavaMail session.
2. Click JavaMail Sessions. You will see the following window, shown in the figure "Configuring a JavaMail Session" in the right pane of the Administration interface:

Figure 10-9 Configuring a JavaMail Session



server1: Java Mail Sessions: New

General

JNDI Name:* NewJavaMailSession

Mail Host:* blr-root.india.sun.com

Default User:* ar126587

Default Return Address:* aruna.r@sun.com

Description: xyz

Java Mail Session Enabled: ☑

3. In the JNDI Name text field, provide a JNDI name for the Java Mail session that you are creating. Once a Java Mail Resource is registered with the server, it can be referenced in any J2EE application component using a JNDI lookup.
4. In the Mail Host text field, specify the DNS name of the default mail server. The connect methods of the Store and Transport objects use this property, to locate the target host, if the protocol-specific host property is absent.

5. In the Default User text field, specify the username to provide when connecting to a Mail server. The connect methods of the Store and Transport objects use this property to obtain the username, if the protocol-specific username property is absent.
6. In the Default Return Address field, specify the default return address of the current user. The default address should be in this form: username@host.
7. In the Description field, provide a description for this Java Mail session.
8. Mark the Java Mail Session Enabled checkbox to enable the Java Mail session that you created.
9. Click OK to save the new JavaMail session that you have configured.

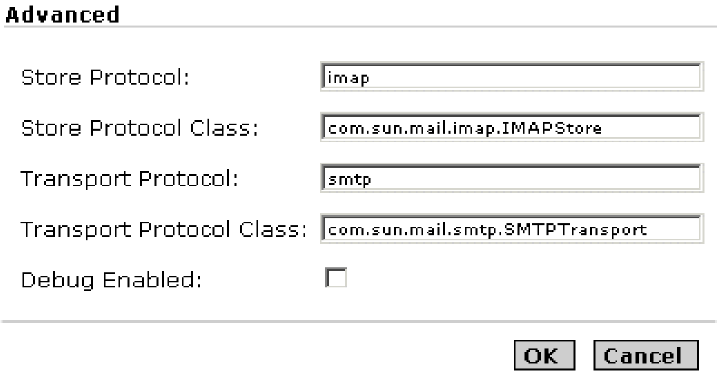
Configuring Advanced Resource Properties

You can configure several additional properties for your new JavaMail session using the Administration interface. The property name and value pairs depend upon the mail protocol in use. You can also directly specify these properties in the server.xml file.

To configure additional properties, perform the following tasks:

1. In left pane of the Administration interface, expand the Sun ONE Application Server instance, whose JavaMail session configuration you want to modify.
2. Click JavaMail Sessions. You will see the following window, shown in the figure "Configuring Additional Resources for JavaMail Session" in the right pane of the Administration interface, below the main configuration section explained in "Creating a New JavaMail Session":

Figure 10-10 Configuring Additional Resources for JavaMail Session



Advanced

Store Protocol: imap

Store Protocol Class: com.sun.mail.imap.IMAPStore

Transport Protocol: smtp

Transport Protocol Class: com.sun.mail.smtp.SMTPTransport

Debug Enabled: □

3. In the Store Protocol text field, specify the store protocol that you would like to use for this specific JavaMail session, for example, POP3, or IMAP.
4. In the Store Protocol Class text field, specify the class name of the store protocol that you have indicated, as shown in the given example.
5. In the Transport Protocol text field, specify the transport protocol that you want to use for this specific session of JavaMail, for example, SMTP.
6. In the Transport Protocol Class text field, specify the class name of the transport protocol you have indicated for this session, as shown in the above example.
7. Mark the Debug Enabled checkbox, to enable debugging of this specific session of JavaMail. Enabling this checkbox turns the debug mode on.
8. Click OK to save your additional properties configuration.

A complete sample of mail resource configuration is shown below:

```
<mail-resource
   jndi-name = "mail/Session"
   enabled = "true"
   store-protocol = "imap"
   store-protocol-class = "com.sun.mail.imap.IMAPStore"
   transport-protocol = "smtp"
   transport-protocol-class = "com.sun.mail.smtp.SMTPTransport"
```

```
    host = "gopostal.acme.com"
    user = "kingkong"
    from = "kingkong@acme.com"
    debug = "false">
    <property name = "mail-imap-host" value = "spaceduck.acme.com"/>
    <property name = "mail-imap-user" value = "fredbloggs"/>
</mail-resource>
```

Chapter 11

Using the JMS Service

Sun ONE Application Server provides support for applications that use the Java Message Service (JMS) application programming interface (API) for messaging operations. JMS is a set of programming interfaces that provide a common way for Java applications to create, send, receive, and read messages in a distributed environment.

In particular, JMS is the standards-based way that Java 2 Enterprise Edition (J2EE) applications perform asynchronous messaging. Accordingly, J2EE components—Web components or Enterprise JavaBeans (EJB) components—can use the JMS API to send messages that can be consumed asynchronously by a specialized EJB, called a message-driven bean (MDB).

Sun ONE Application Server support for JMS messaging, in general, and for MDBs, in particular, requires messaging middleware that implements the JMS specification—a JMS provider. Sun ONE Application Server uses Sun ONE Message Queue (MQ), Version 3.01, as its native JMS provider.

MQ is tightly integrated into Sun ONE Application Server, providing transparent JMS messaging support. This support (known within Sun ONE Application Server as the *JMS Service*) requires only minimal administration.

This chapter provides information needed to understand and administer the built-in JMS Service provided through Sun ONE Message Queue. It includes the following topics:

- About JMS
- The Built-in JMS Service
- Administration of the Built-in JMS Service

About JMS

The JMS specification describes a set of programming interfaces that support distributed, enterprise messaging. An enterprise messaging systems enables independent distributed components or applications to interact through messages. These components, whether on the same system, the same network, or loosely connected through the Internet, use messaging to pass data and coordinate their respective functions.

To support enterprise-scale messaging, JMS provides for reliable, asynchronous message delivery.

Reliable delivery. Messages from one component to another must not be lost due to network or system failure. This means the system must be able to guarantee that a message is successfully delivered.

Asynchronous delivery. For large numbers of components to be able to exchange messages simultaneously, and support high density throughputs, the sending of a message cannot depend upon the readiness of the consumer to immediately receive it. If a consumer is busy or offline, the system must allow for a message to be sent and subsequently received when the consumer is ready. This is known as asynchronous message delivery, popularly known as store-and-forward messaging.

This topic is a brief overview of JMS concepts and terminology:

- Basic Messaging System Concepts
- The JMS Specification
- Message-driven Beans

For a full description of JMS, see the JMS 1.0.2 specification, which can be found at the following location:

```
http://java.sun.com/products/jms/docs.html
```

Basic Messaging System Concepts

A few basic concepts underlie enterprise messaging systems in general, and JMS in particular. These are discussed in this topic.

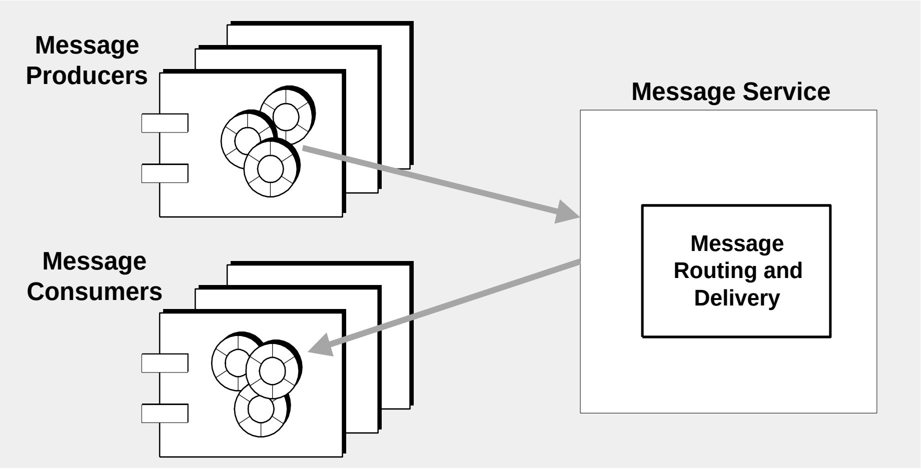
Message

A message consists of data in some format (message body) and meta-data that describes the characteristics or properties of the message (message header), such as its destination, lifetime, or other characteristics determined by the messaging system.

Message Service Architecture

The basic architecture of a messaging system is illustrated in the figure “Message Service Architecture.” It consists of message producers and message consumers that exchange messages by way of a common message service. In general, any number of message producers and consumers can reside in the same messaging component. A message producer sends a message to a message service. The message service, in turn, using message routing and delivery components, delivers the message to one or more message consumers that have registered an interest in the message. The message routing and delivery components are responsible for guaranteeing delivery of the message to all appropriate consumers.

Figure 11-1 Message Service Architecture



Message Delivery Models

There are many relationships between producers and consumers: one to one, one to many, and many to many relationships. For example, you might have messages delivered from:

- one producer to one consumer

- one producer to many consumers
- many producers to one consumer
- many producers to many consumers.

These relationships are often reduced to two message delivery models: *point-to-point* and *publish/subscribe* messaging. The focus of the point-to-point delivery model is on messages that originate from a specific producer and are received by a specific consumer. The focus of publish / subscribe delivery model is on messages that originate from any of a number of producers and are received by any number of consumers. These message delivery models can overlap.

Historically, messaging systems supported various combinations of these two delivery models. The JMS API was intended to create a common programming approach that supports both point to point and publish / subscribe delivery models.

The JMS Specification

JMS specifies a message structure, a programming model, and a set of rules and semantics that govern messaging operations.

JMS Message Structure

According to the JMS specification, a message is composed of three parts: a header, properties (which can be thought of as an extension of the header), and a body.

Header. The header specifies the JMS characteristics of the message: its destination, whether is persistent or not, its time to live, and its priority. These characteristics govern how the messaging system delivers the message.

Properties. Properties are optional—they provide values that applications can use to filter messages according to various selection criteria. Properties are optional.

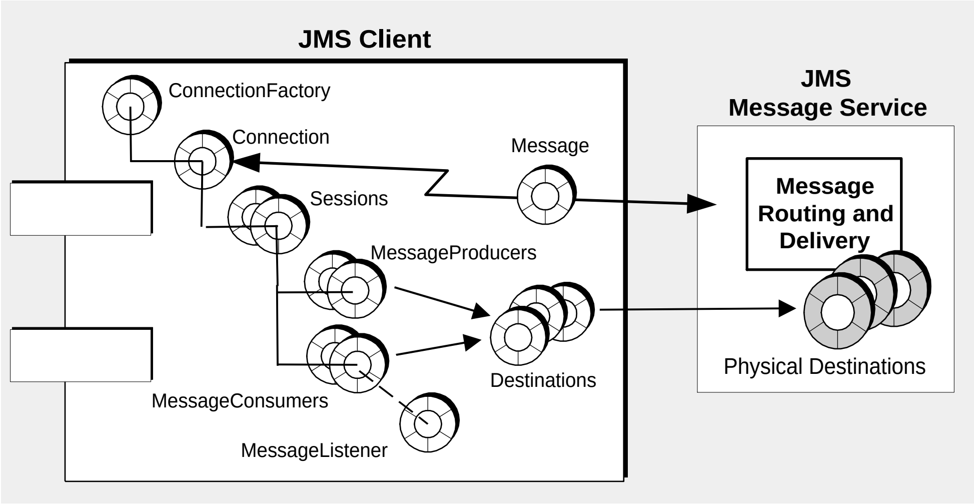
Message body. The message body contains the actual data to be exchanged. JMS supports six body types.

JMS Programming Model

In the JMS programming model, JMS clients (components or applications) exchange messages by way of a JMS message service. Message producers send messages to the message service, from which message consumers receive them. These messaging operations are performed using a set of objects (furnished by a JMS provider) that implement the JMS API. The figure "JMS Programming Objects" shows the JMS objects used to program the delivery of messages.

In the JMS programming model, a JMS client uses a ConnectionFactory object to create a connection over which messages are sent to and received from the JMS message service. A Connection is a JMS client's active connection to the message service. Both allocation of communication resources and authentication of the client take place when a connection is created.

Figure 11-2 JMS Programming Objects



The connection is used to create sessions. A Session is a single-threaded context for producing and consuming messages. It is used to create the message producers and consumers that send and receive messages. A session supports reliable delivery through a number of acknowledgement options or through transactions (which can be managed by a distributed transaction manager).

A JMS client uses a MessageProducer to send messages to a specified physical destination, represented in the API as a destination object. The message producer can specify a default delivery mode (persistent vs. non-persistent messages), priority, and time-to-live that govern all messages sent by the producer to the physical destination.

Similarly, a JMS client uses a MessageConsumer to receive messages from a specified physical destination, represented in the API as a destination object. A message consumer can support either synchronous or asynchronous consumption of messages. Asynchronous consumption is achieved by registering a MessageListener with the consumer. The client consumes a message when a session thread invokes the onMessage() method of the MessageListener object.

Administered Objects: Provider Independence

Two of the objects described in the figure "JMS Programming Model" on page 302 depend on the details of how a JMS provider implements a JMS message service. The connection factory object depends on the underlying protocols and mechanisms used by the provider to deliver messages, and the destination object depends on the specific naming conventions and capabilities of the physical destinations used by the provider.

Normally these provider-specific characteristics would make JMS client code dependent on the details of a JMS API implementation. To make JMS client code provider-independent, however, the JMS specification requires that provider-specific objects—called administered objects—be accessed in a standardized way, rather than directly instantiated in client code.

Administered objects encapsulate provider-specific implementation and configuration information. They are created and configured by an administrator, stored in a name service, and accessed by client applications through standard JNDI lookup code. Using administered objects in this way makes JMS client code provider-independent.

JMS provides for two general types of administered objects: connection factories and destinations. Both encapsulate provider-specific information, but they have very different uses within a JMS client. A connection factory is used to create connections to a message server, while destination objects are used to identify physical destinations used by the JMS message service.

NOTE In the context of the Sun ONE Application Server, JMS administered objects are regarded as JMS resources, similar to other Application Server resources.

Message-driven Beans

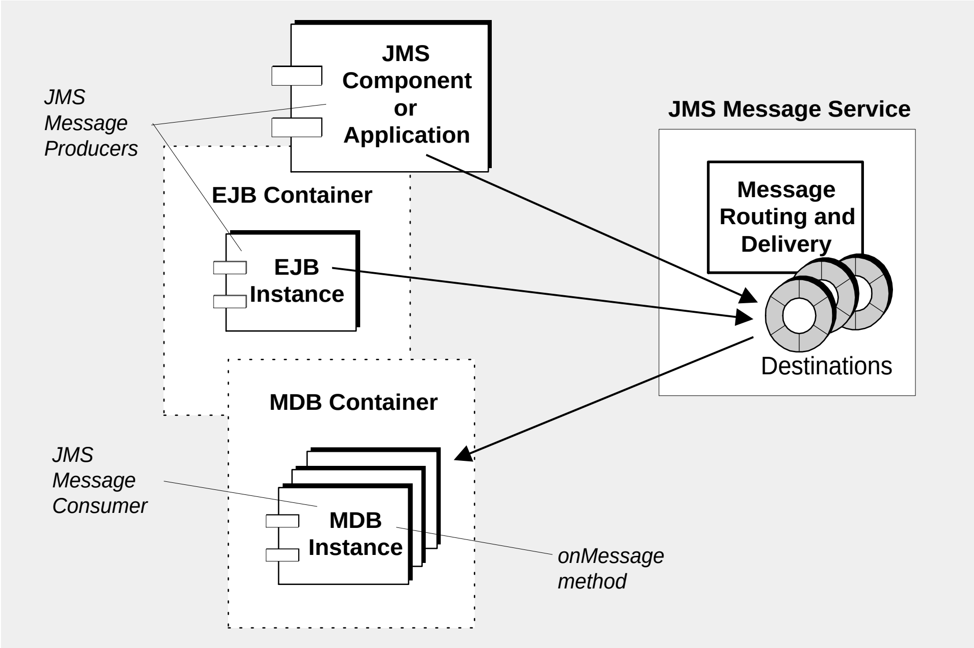
In addition to the general JMS client programming model introduced in the figure "JMS Programming Model" on page 302, there is a more specialized adaptation of the JMS API used in the context of J2EE applications. This specialized JMS client is called a *message-driven bean* and is one of a family of EJB components specified in the EJB 2.0 Specification (`http://java.sun.com/products/ejb/docs.html`).

The need for message-driven beans arises out of the fact that other EJB components (session beans and entity beans) can only be called synchronously. Hence, when you invoke a method of these beans, resources are blocked until the methods have completed. These EJB components have no mechanism for receiving messages asynchronously, since they are only accessed through standard EJB interfaces.

However, asynchronous messaging is a requirement of many enterprise applications. Hence, the need for an EJB component that can receive messages and consume them without being tightly coupled to the producer of the message.

The MDB is a specialized EJB component supported by a specialized EJB container (a software environment that provides distributed services for the components it supports).

Figure 11-3 MDB Message Consumers



Message-driven Bean. The MDB is a JMS message consumer that implement the JMS `MessageListener` interface. Its `onMessage` method (written by the MDB developer) is invoked when a message is received by the MDB container. The `onMessage` method consumes the message, just as the `onMessage` method of any

JMS MessageListener object would. The MDB can consume messages from a single destination. The messages can be produced by standalone JMS client applications, Web components, or other EJB components, as shown in the "MDB Message Consumers" figure.

MDB Container. The MDB is supported by a specialized EJB container, responsible for creating instances of the MDB and setting them up for asynchronous consumption of messages. This involves setting up a connection with the message service (including authentication), creating a pool of sessions associated with a given destination, and managing the distribution of messages as they are received among the pool of sessions and associated MDB instances. Since the container controls the life-cycle of MDB instances, it manages the pool of MDB instances so as to accommodate incoming message loads.

Associated with an MDB is a deployment descriptor that specifies the JNDI lookup names for the administered objects used by the container in setting up message consumption: a connection factory and a destination. The deployment descriptor might also include other information that can be used by deployment tools to configure the container. Each such container supports instances of only a single MDB.

For information on configuring an EJB container's MDB settings for the Sun ONE Application Server, see "About Message-driven Beans" on page 213.

The Built-in JMS Service

Support for JMS messaging, in general, and for MDBs, in particular, is built into Sun ONE Application Server. This support is achieved through the tight integration of Sun ONE Message Queue with the Sun ONE Application Server, providing a native, built-in JMS Service.

This topic covers the following topics necessary to understand this built-in JMS Service:

- About Sun ONE Message Queue (MQ)
- Integration of MQ with Sun ONE Application Server

For information on administering the built-in JMS Service, see "Administration of the Built-in JMS Service" on page 314.

About Sun ONE Message Queue (MQ)

Sun ONE Message Queue (MQ) is an enterprise messaging system that implements the JMS open standard: it is a JMS provider.

The MQ product has features which go beyond the minimum requirements of the JMS specification for reliable, asynchronous messaging. Some of these features (centralized administration, tunable performance, support for multiple messaging transports. user authentication and authorization) are available in the MQ Platform Edition integrated into Sun ONE Application Server. Additional features (scalable message servers and secure messaging) are available by upgrading to the MQ Enterprise Edition.

An MQ messaging system consists of a number of parts, as illustrated in the figure "MQ System Architecture" on page 308, that work together to provide for reliable message delivery.

The main parts of an MQ messaging system are the following:

- MQ Message Server
- MQ Client Runtime
- MQ Administered Objects
- MQ Administration Tools

These are introduced briefly in the following topics. For a more complete discussion of the MQ messaging system, see the MQ *Administrator's Guide*, which can be found at the following location:

```
http://docs.sun.com/
```

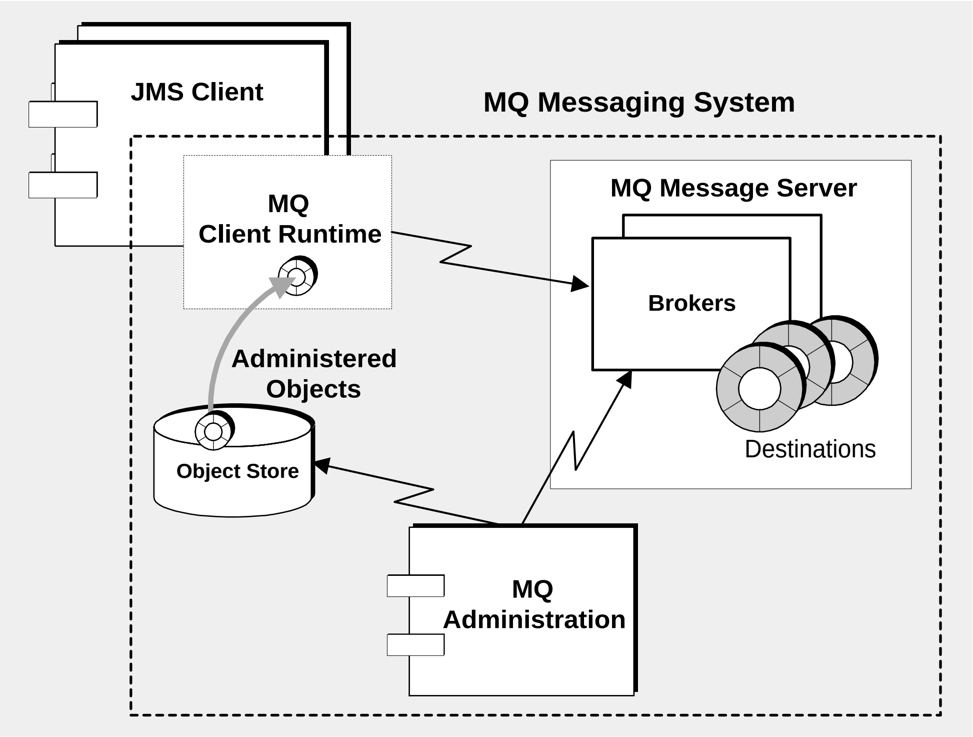
MQ Message Server

The main parts of an MQ message server, as shown in the figure "MQ System Architecture" on page 308, are brokers and physical destinations.

Brokers. A broker provides delivery services for an MQ messaging system. Message delivery relies upon a number of supporting components that handle connection services, message routing and delivery, persistence, security, and logging. A message server can employ one or more brokers to achieve scalability.

Physical Destinations. Delivery of a message is a two-phase process—delivery from a producing client to a physical destination maintained by a broker, followed by delivery from the destination to one or more consuming clients. Physical destinations represent locations in a broker's physical memory and/or persistent storage (see "Physical Destinations" on page 309 for more information).

Figure 11-4 MQ System Architecture



Brokers

Message delivery in an MQ messaging system—from producing clients to destinations, and then from destinations to one or more consuming clients—is performed by a broker (or, in the MQ 3.01 Enterprise Edition, a cluster of brokers working in tandem). To perform message delivery, a broker must set up communication channels with clients, perform authentication and authorization, route messages appropriately, guarantee reliable delivery, and provide data for monitoring system performance.

To perform this complex set of functions, a broker uses a number of different components, each with a specific role in the delivery process. You can configure these internal components to optimize the performance of the broker, depending on load conditions, application complexity, and so on. See the MQ *Administrator's Guide* for more information.

Physical Destinations

MQ messaging is premised on a two-phase delivery of messages: first, delivery of a message from a producer client to a destination on the broker, and second, delivery of the message from the destination on the broker to one or more consumer clients. There are two types of destinations: queues (point to point delivery model) and topics (publish / subscribe delivery model). These destinations represent locations in a broker's physical memory where incoming messages are marshaled before being routed to consumer clients.

You generally create physical destinations using administration tools, however destinations can also be automatically created by the broker upon receipt of a message.

Queue Destinations. Queue destinations are used in point to point messaging, where a message is meant for ultimate delivery to only one of a number of consumers that has registered an interest in the destination. As messages arrive from producer clients, they are queued and delivered to a consumer client.

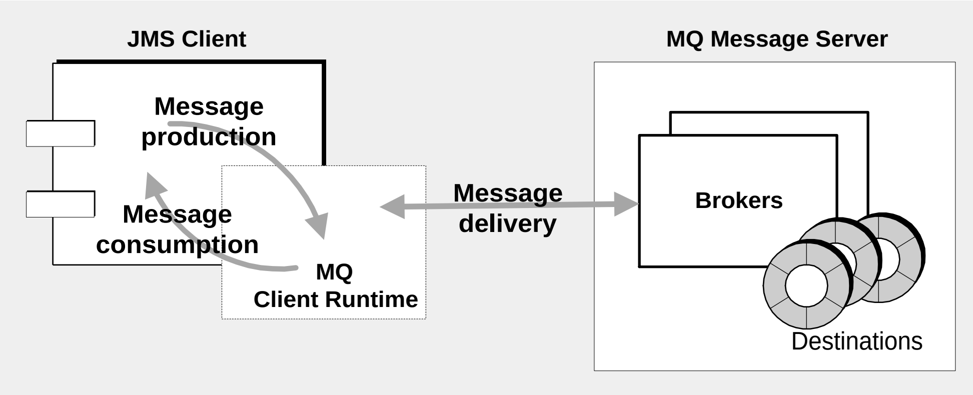
Topic Destinations. Topic destinations are used in publish / subscribe messaging, where a message is meant for ultimate delivery to all of the consumers that have registered an interest in the destination. As messages arrive from producers, they are routed to all consumers subscribed to the topic. If consumers have registered a durable subscription to the topic, they do not have to be active at the time the message is delivered to the topic—the broker will store the message until the consumer is once again active, and then deliver the message.

MQ Client Runtime

The MQ client runtime provides JMS clients (stand-alone applications, Web components or EJB components) with an interface to the MQ message server—it supplies JMS clients with implementations of all the programming interfaces needed for clients to send messages to destinations and to receive messages from such destinations.

The figure "Messaging Operations" on page 310 illustrates how message production and consumption involve an interaction between JMS clients and the MQ client runtime, while message delivery involves an interaction between the MQ client runtime and the MQ message server.

Figure 11-5 Messaging Operations



MQ Administered Objects

MQ administered objects allow JMS client code to be provider-independent (see "Administered Objects: Provider Independence" on page 304). They do this by encapsulating provider-specific implementation and configuration information in objects that can then be used by client applications in a provider-independent way. MQ administered objects are created and configured by an administrator, stored in a name service, and accessed by JMS clients through standard JNDI lookup code.

Connection Factory Administered Objects. A connection factory object is used to create physical connections between a JMS client (stand-alone applications, Web components or EJB components) and an MQ message server. A connection factory object has no physical representation in a broker—it is used simply to enable a JMS client to establish connections with a broker. It is also used to specify behaviors of the connection and of the client runtime that uses the connection to access a broker: an MQ connection factory therefore has a significant number of configurable attributes that allow you to tune the performance of an MQ system.

Destination Administered Objects. A destination administered object (a queue or a topic) represents a physical destination (a physical queue or a physical topic) in a broker to which the publicly-named destination administered object corresponds. By creating a destination administered object, you allow JMS clients (message consumers and/or message producers) to access the corresponding physical destination.

MQ Administration Tools

MQ administration tools fall into two categories: command line utilities and a graphical user interface (GUI) Administration Console.

The Administration Console. You can use the administration console to connect to a broker and manage it, create physical destinations on the broker, connect to an object store and add, update, or delete administered objects from the object store. There are some tasks that you cannot use the Administration Console to perform; chief among these are starting up a broker, creating broker clusters, configuring more specialized properties of a broker, and managing a user database.

Command Line Utilities. You use the MQ utilities to perform all the tasks you can perform using the administration console, and in addition, to start up and manage a broker, configure more specialized properties of a broker, and manage a MQ user database.

Integration of MQ with Sun ONE Application Server

MQ Platform Edition is automatically installed as part of the Sun ONE Application Server installation process. For more information, see the *Sun ONE Application Server Installation Guide.*

This installation provides Sun ONE Application Server with a JMS messaging system that supports any number of Sun ONE Application Server instances. Each server instance, by default, has an associated built-in JMS Service that supports all JMS clients running in the instance.

This topic covers the following topics:

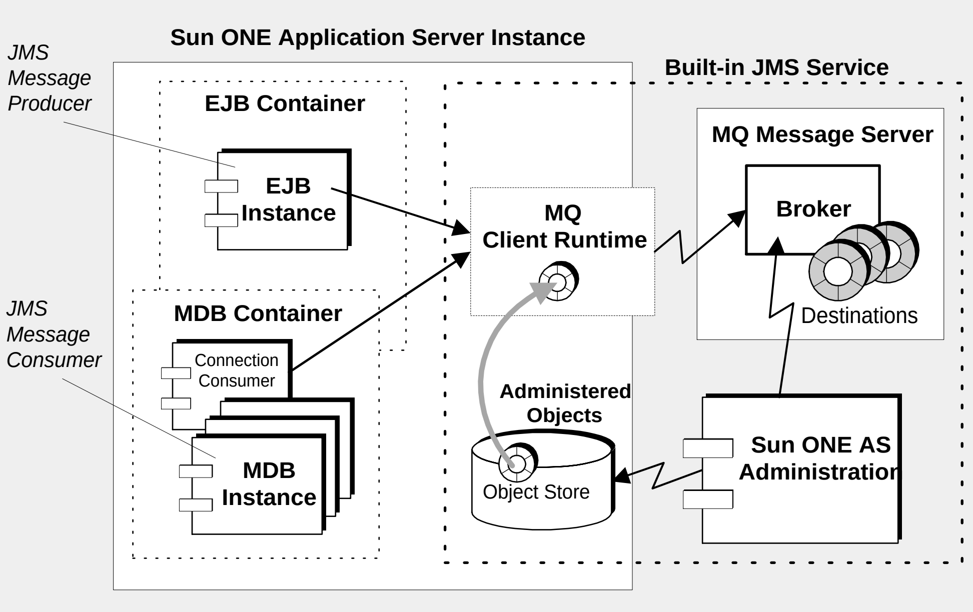
- Architecture of the Built-in JMS Service
- Disabling the Built-in JMS Service

The built-in JMS Service can be administered using Sun ONE Application Server administration tools (see "Administration of the Built-in JMS Service" on page 314).

Architecture of the Built-in JMS Service

The built-in JMS Service—illustrated in the figure "Built-in MQ Messaging System" on page 312—is similar to an ordinary MQ messaging system (shown in the figure "MQ System Architecture" on page 308), except for a number of qualifications, described below.

Figure 11-6 Built-in MQ Messaging System



MQ Message Server. Each Sun ONE Application Server instance is associated with its own built-in JMS Service. The built-in JMS Service makes use of a single-broker message server. The broker runs in a separate process outside of the Sun ONE Application Server instance as shown in the figure "Built-in MQ Messaging System." By default, the broker instance (the built-in JMS Service) starts up when its associated server instance is started up and shuts down when the server instance is shut down. Configuration information for a server instance's built-in JMS Service is recorded in the Sun ONE Application Server configuration store (the `server.xml` file) and can be modified as described in "Configuring the JMS Service" on page 315.

MQ Client Runtime. The client runtime part of the JMS Service is a set of libraries that support the JMS API. Any JMS clients (JMS client components, including MDBs) that run in a server instance have access to these libraries.

MQ Administered Objects. The built-in JMS Service uses an object store provided by Sun ONE Application Server. Each server instance has its own object store. The JMS Service stores administered objects (connection factory and destination resources) in this object store. You create these administered object resources as described in "Managing Administered Object Resources" on page 321, and they are accessed by JMS clients using JNDI lookup code.

Sun ONE Application Server Administration. The Sun ONE Application Server Administration interface and command line utilities implement a limited subset of MQ administration capability. The Administration interface and command line let you configure the built-in JMS Service, create and delete physical destinations, and create and delete administered object resources needed by JMS clients to perform JMS messaging operations. These administration tools, however, do not allow (or do not facilitate) performing more sophisticated administration tasks, such as setting broker properties, tuning the MQ client runtime, modifying the MQ user repository, managing MQ security, and so forth. If you wish to perform these administration tasks for the built-in JMS Service, you have to use the administration tools installed with MQ and described in the MQ *Administrator's Guide*. For a comparison of MQ and Sun ONE Application Server administration capability, see the table "Comparison of Sun ONE Message Queue and Sun ONE Application Server Administration Capability" on page 315.

Disabling the Built-in JMS Service

By default, the built-in JMS Service is started up (that is, the MQ broker is started up) when the associated Sun ONE Application Server instance is started up. However, you might want to disable the automatic startup of the JMS Service when you start up a server instance, either because the server instance does not need to support JMS messaging or because the server instance uses an *external JMS Service*. (To disable the built-in JMS Service, see "Configuring the JMS Service" on page 315.)

An external JMS Service is a messaging system that is not controlled from within Sun ONE Application Server. In the case of MQ—the *native* JMS provider—this means that you simply start up and manage your MQ message server independently, using MQ administration tools. JMS clients running in various server instances can still use MQ administered objects to access the MQ message server. These administered objects can either be stored in the object store associated with each application server instance, or in a separate, independent object store managed by the MQ administration tools (where the store can be shared by multiple server instances if necessary).

There are a number of scenarios in which a server instance might use an external JMS Service. The most likely is when JMS clients in different server instances need to access the same physical destinations. In that case, the server instances must all access the same message server. To achieve this, you disable the built-in JMS Service for all server instances and configure all JMS clients to perform the appropriate JNDI lookups to access the external JMS Service. In addition, you independently administer the external JMS Service (managing the message server, creating physical destinations, and creating all needed administered objects) using the external JMS Service provider's administration tools.

To configure multiple application server instances to share a single MQ broker instance:

1. Disable the JMS service on all the server instances.
2. Manage the shared MQ broker independently of any server instance, which means you must start up and shut down the broker using the administration tools for managing the external service. You must also manage physical destinations independently of the Sun ONE Application Server.
3. Configure a connection factory JMS resource in each server instance such that it points to this external MQ broker (you must set the `imqBrokerHostName` and `imqBrokerHostPort` properties appropriately).
4. Use this connection factory resource when deploying JMS applications on the Sun ONE Application Server.

It is possible to have both an external and built-in JMS Service running at the same time. JMS clients in a server instance can access whichever JMS Service they need.

One possibility that is *not* recommended is to have a number of server instances share the same built-in JMS Service (one is enabled, the rest are disabled). This is not recommended because the enabled JMS Service only runs when its associated server instance runs, making it very difficult to manage the situation.

When you disable a built-in JMS Service, you also disable your ability to perform administrative tasks associated with that built-in JMS Service. All administrative tasks needed to support an external JMS Service must be performed using the administration tools for managing that external service.

Administration of the Built-in JMS Service

This topic focuses on administration of the built-in JMS Service. *Administration is performed on a server instance-by-server instance basis.*

Administration of the built-in JMS Service involves the following tasks:

- Configuring the JMS Service
- Managing Physical Destinations
- Managing Administered Object Resources
- Administering the Built-in JMS Service Using the Command-Line Interface

Administration can be performed using the Sun ONE Application Server's Administration interface or command line utility. A comparison of these administration tools with MQ administration tools is shown in the table "Comparison of Sun ONE Message Queue and Sun ONE Application Server Administration Capability."

Table 11-1 Comparison of Sun ONE Message Queue and Sun ONE Application Server Administration Capability

Function	Sun ONE MQ Administration Tools	Sun ONE AS Administration interface	Sun ONE AS Administration Command Line
Manage MQ broker state	yes	start / stop	start / stop
Configure MQ broker	yes	no	no
Manage MQ broker user repository	yes	no	no
Multi-broker clusters	yes	no	no
Manage security	yes	no	no
Manage physical destinations	yes	create / delete	create / delete
Manage durable subscriptions and transactions	yes	no	no
Manage administered object resources	yes	create / delete / configure	yes

The following sections explain how to perform JMS Service administration tasks using the Sun ONE Application Server Administration interface.

Configuring the JMS Service

At installation time a number of JMS Service properties are set for the built-in JMS Service. You can change the default values of these properties by configuring the JMS Service.

JMS Service properties are described in the table "JMS Service Properties."

Table 11-2 JMS Service Properties

Property	Description	Default Value
Log level	The level of logging information you want written to the Sun ONE Application Server log file. For more information, see Chapter 5, “Using Logging.”	`DEBUG_HIGH`
Port	The primary port number of the broker instance providing built-in JMS Service. By default, the built-in JMS Service uses the default primary port number. However, if that port conflicts with other software, or if you are starting up more than one Sun ONE Application Server instance, you need to specify a unique primary port number for each. Note that it is possible to have a port conflict if the port number assigned to the JMS Service at installation time later is used by another service. In this case, you have to give the JMS Service a different port number.	`7676`
Administrator’s username/password	The username/password needed to perform broker administration tasks, such as managing physical destinations (see “Managing Physical Destinations” on page 318). If you want administrator access to a broker instance to be secure (by default, any user is provided access), you need to first make the appropriate entries in the broker’s user repository (as described in the MQ Administrator’s Guide) and then enter the corresponding values here for the administrator’s username and password.	`admin/admin`
Start Timeout	Specifies the time in seconds that the server instance waits for the JMS Service to start up. If this timeout is exceeded, the server instance startup is aborted.	`60`
Start Arguments	Specifies any arguments that will be used in starting up the JMS Service. The startup arguments for the `imqbroker` command, and how to specify them, can be found in the MQ *Administrator’s Guide*. (The `-name` and `-port` arguments, if supplied, are ignored.)	

Table 11-2 JMS Service Properties *(Continued)*

Property	Description	Default Value
Startup Enabled	Specifies whether the built-in JMS Service is started up when the server instance starts up. If you are not supporting JMS messaging or are using an external JMS message service, set this property to FALSE.	TRUE

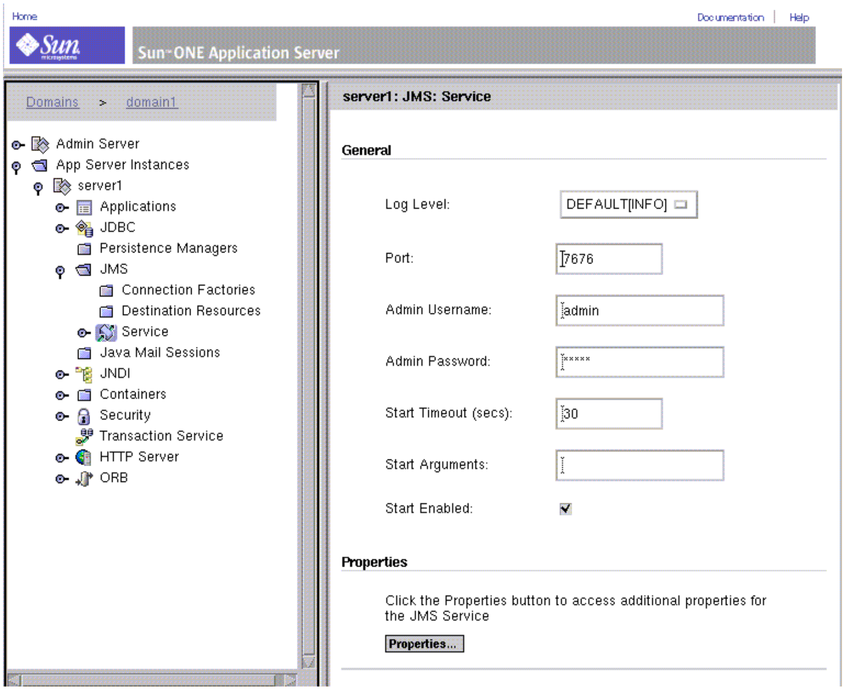
The built-in JMS Service can be configured before starting up its corresponding server instance. If the server instance is already running, however, configuration changes take effect only after the server instance is stopped and subsequently restarted.

To configure the built-in JMS Service:

1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Select the Service link.

   The JMS Service configuration screen is displayed on the right pane.

Figure 11-7 JMS Service Configuration Screen



server1: JMS: Service

General

Log Level: DEFAULT[INFO]

Port: 7676

Admin Username: admin

Admin Password: *****

Start Timeout (secs): 30

Start Arguments:

Start Enabled: ☑

Properties

Click the Properties button to access additional properties for the JMS Service

5. Modify any of the properties you wish (see the table "JMS Service Properties" on page 316).

6. Click the Save button.

   The JMS Service screen is refreshed.

Managing Physical Destinations

In JMS messaging, a JMS producer sends messages to a physical destination on a message service from which they are dispatched to a JMS consumer.

For the built-in JMS Service, you can create these physical destinations explicitly or they can be created automatically by the JMS Service (MQ broker) upon receipt of a message. In general, you have more control over the messaging system and its resources by explicitly creating the physical destinations needed by messaging applications. When these destinations are no longer needed, you can delete them.

In order to create or delete physical destinations for a built-in JMS Service, the JMS Service must be running and the administrator username and password (specified when you configure the built-in JMS Service) must correspond to valid entries in the broker's user repository (see the table "JMS Service Properties" on page 316).

Using the Administration interface, you can perform the following management tasks for physical destinations on a built-in JMS Service:

- Create a Queue or Topic Destination
- List Physical Destinations
- Delete a Physical Destination

Create a Queue or Topic Destination

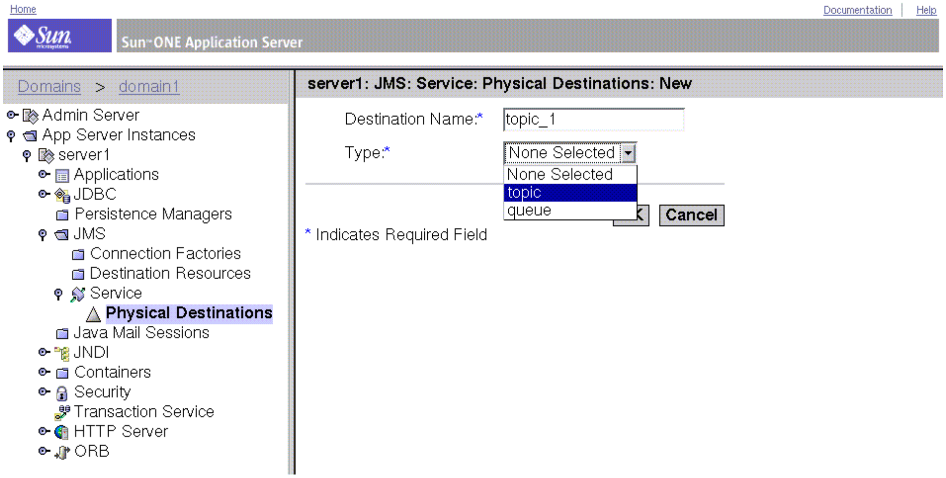
To create a queue or topic destination:

1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Open the Service link.
5. Select the Physical Destinations link

   The Physical Destinations screen is displayed in the right pane.
6. Click the New button.

   The Physical Destinations: New screen is displayed in the right pane.

Figure 11-8 New JMS Physical Destination Screen



server1: JMS: Service: Physical Destinations: New

Destination Name:* topic_1

Type:* None Selected

* Indicates Required Field

7. Enter the name of the physical destination.

8. Select queue or topic from the Type pull down.

9. Click the OK button.

   The right pane is refreshed and displays the new queue or topic destination in the list of existing queue and topic destinations.

List Physical Destinations

To list existing queue and topic destinations:

1. Open the Administration interface.

2. Open a server instance in the left pane.

3. Open the JMS folder.

4. Open the Service link.

5. Select the Physical Destinations link

   The right pane displays the current physical destinations.

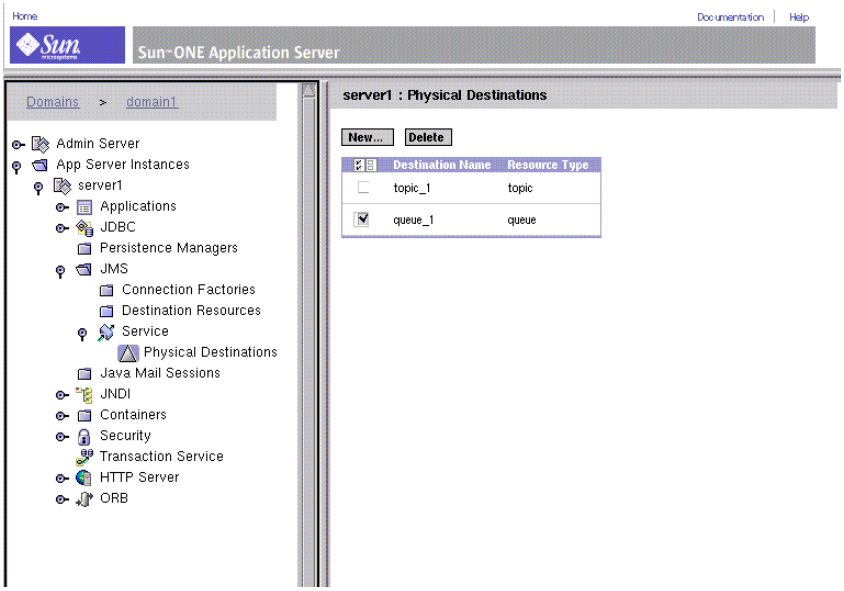
Delete a Physical Destination

You can delete queue or topic destinations as needed.

To delete a physical destination:

1. List the existing destinations (see "List Physical Destinations" on page 320).
2. Click within the Select box for each destination you wish to delete.

Figure 11-9 JMS Physical Destinations Screen



	Destination Name	Resource Type
□	topic_1	topic
☑	queue_1	queue

3. Click the Delete button to remove the selected destinations.

   The list refreshes, showing the remaining destinations.

Managing Administered Object Resources

MQ administered objects, regarded by Sun ONE Application Service as *JMS Resources*, are used by JMS clients to access a JMS Service (either built-in or external).

Two administered object resources—connection factories and destinations—are used by J2EE components to obtain connections to a JMS Service, and then to deliver messages to and from physical destinations on the Service (see "MQ Administered Objects" on page 310).

Because the creation of administered object resources does not involve the JMS Service directly, you do not need to enable the JMS Service, and you do not need a valid username and password (see the table "JMS Service Properties" on page 316), to create administered object resources for a server instance.

Administered Object Attributes

To support JMS messaging, you must create the administered object resources required by all JMS clients running in a server instance. At a minimum, you need to specify a JNDI lookup name for each administered object resource, its type (connection factory, queue, or topic), a description (optional) and whether or not the resource is enabled. Other attributes are discussed in the following sections.

Destination (queue or topic)

In the case of queue or topic administered objects, you also need to specify the name of the corresponding physical destination.

Connection Factory

In the case of connection factory administered objects, the Administration interface creates, by default, a connection factory that points to the built-in JMS Service—that is, to a broker instance whose hostname is the local host and whose port number is that set when configuring the JMS Service (see the table "JMS Service Properties" on page 316).

However, if you disable the JMS Service for a particular server instance, then any JMS clients supported by that server instance must use an external JMS Service. When you create a connection factory to be used to create connections to that external JMS Service, you need to set attributes that specify the hostname and port number of the appropriate broker instance.

Connection factory administered objects have additional attributes used to tune the server instance's MQ client runtime. These are documented in the MQ *Developer's Guide.*

Connection factory administered objects created using the Administration interface support distributed transaction managers.

Administered Object Resource Management Tasks

From the Administration interface, you can perform the following management tasks for administered object resources:

- Create a Queue or Topic Administered Object (Destination Resource)
- Create a ConnectionFactory Administered Object

- List Administered Object Resources
- Delete an Administered Object Resource

Create a Queue or Topic Administered Object (Destination Resource)

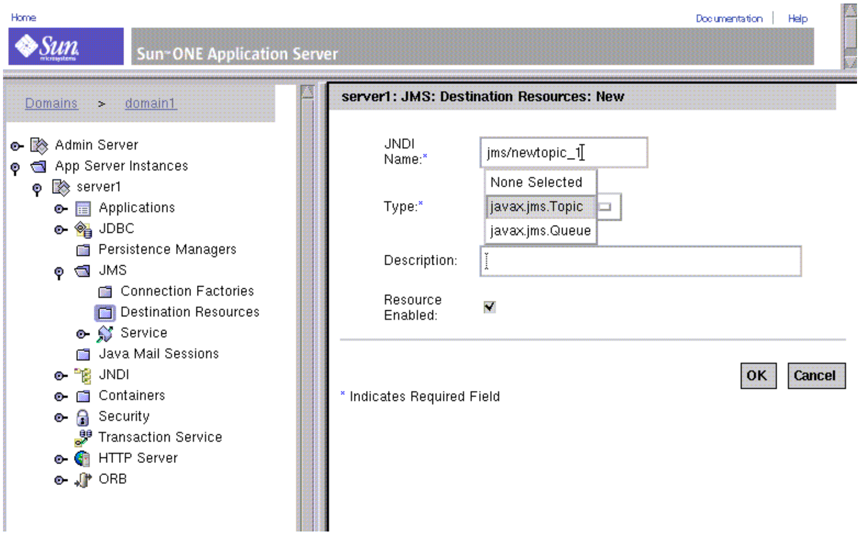
To create a queue or topic administered object:

1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Select the Destination Resources link.

   The Destination Resources screen is displayed in the right pane.
5. Click on the New button.

   The Destination Resources: New screen is displayed.

Figure 11-10 New Destination Administered Object Screen



server1: JMS: Destination Resources: New

JNDI Name:* jms/newtopic_1

Type:* None Selected / javax.jms.Topic / javax.jms.Queue

Description:

Resource Enabled: ☑

* Indicates Required Field

6. Enter the JNDI lookup name associated with this destination administered object.
7. Select the “queue” or “topic” object type from the pull down list.

8. Click the OK button.

   The right pane re-displays the Destination Resource: New screen.

In addition, you must specify the destination name for the object by specifying the `imqDestinationName` property. The value of this property should match the physical destination's name.

To change the value of this property:

1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Open the Destination Resources folder.
5. Select the Destination Resource you want to edit.

   The right pane displays the Destination Resource screen.

6. In the right pane, click Properties.

   The Edit Properties screen appears.

7. In the Name field, enter imqDestinationName.
8. In the Value field, enter the physical destination's name.
9. Click OK.

   The right pane redisplays the Destination Resources screen.

Create a ConnectionFactory Administered Object

To create a queue connection factory or topic connection factory administered object:

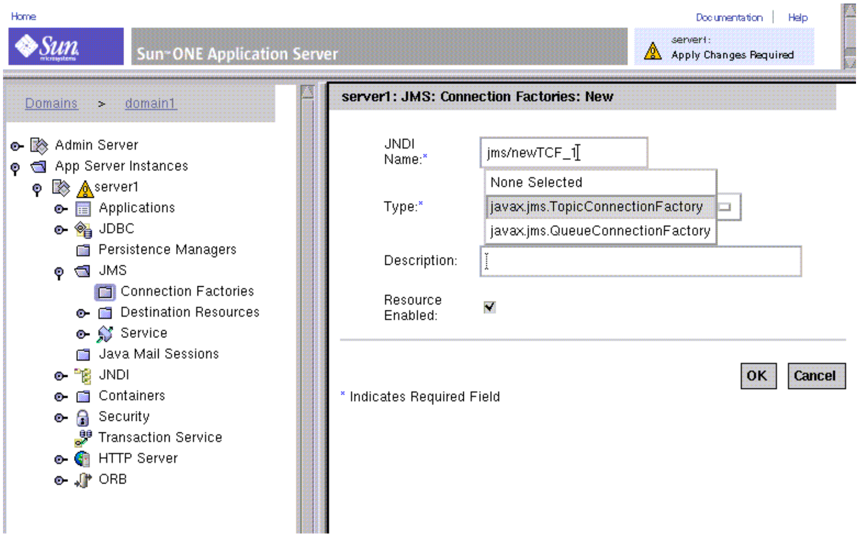
1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Select the Connection Factories link.

   The Connection Factory Resources screen is displayed in the right pane.

5. Click on the New button.

   The Connection Factory Resources: New screen is displayed.

Figure 11-11 New ConnectionFactory Administered Object Screen



server1: JMS: Connection Factories: New

JNDI Name:* jms/newTCF_1

Type:* None Selected / javax.jms.TopicConnectionFactory / javax.jms.QueueConnectionFactory

Description:

Resource Enabled: ☑

* Indicates Required Field

6. Enter the JNDI lookup name associated with this connection factory administered object.
7. Select the connection factory object type from the pull down list.
8. Click the OK button.

   The right pane re-displays the Connection Factory Resources screen with the newly created connection factory object in the list.

If this connection factory creates connections to a broker other than that of the built-in JMS Service, you must set values for the `imqBrokerHostName` and `imqBrokerHostPort` properties.

To change the value of these properties:

1. Open the Administration interface.
2. Open a server instance in the left pane.
3. Open the JMS folder.
4. Open the Connection Factories folder
5. Select the Connection Factory resource you want to edit.

   The right pane displays the Connection Factory screen.

6. In the right pane, click Properties.

   The Edit Properties screen appears.

7. In the Name field, enter `imqBrokerHostName`.

8. In the Value field, enter a value for the property.

9. In the Name field and `imqBrokerHostPort`.

10. In the Value field, enter a value for the property.

11. Click OK.

    The right pane redisplays the Connection Factory screen.

List Administered Object Resources

To list existing administered objects:

1. Open the Administration interface.

2. Open a server instance in the left pane.

3. Open the JMS folder.

4. Select the Destination Resources or Connection Factory Resources link

   The right pane displays the current destination or connection factory administered objects.

Delete an Administered Object Resource

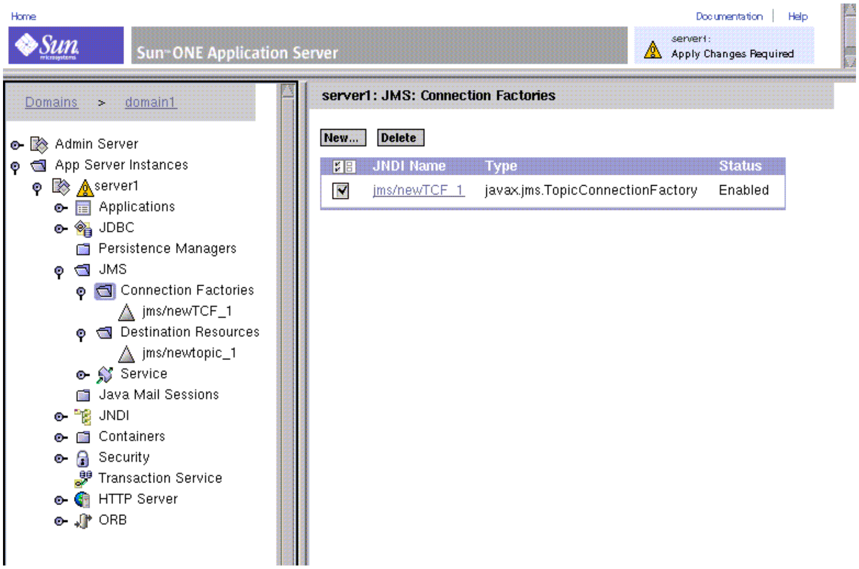
To delete an administered object resource:

1. List the existing administered object resources (see "List Administered Object Resources" on page 326).

   The right pane displays the existing administered object resources.

2. Click within the Select box for each object you wish to delete.

Figure 11-12 JMS Connection Factory Screen, Refreshed



server1: JMS: Connection Factories

☐	JNDI Name	Type	Status
☑	jms/newTCF_1	javax.jms.TopicConnectionFactory	Enabled

3. Click the Delete button to remove each selected object.

   The screen refreshes the list, showing the remaining administered object resources.

Administering the Built-in JMS Service Using the Command-Line Interface

The Sun ONE Application Server has a command-line utility, `asadmin`, which you can use to perform all the same tasks as you can perform with the Administration interface.

Use the following asadmin commands to configure and administer the built-in JMS Service.

asadmin Commands for Administering the Built-in JMS Service

Command	Use
add-resources	Adds one or more resources of type jdbc, jms, or javamail.
create-jmsdest	Creates a JMS physical destination.

asadmin Commands for Administering the Built-in JMS Service

Command	Use
create-jms-resource	Creates a JMS resource.
delete-jmsdest	Deletes a JMS physical destination.
delete-jms-resource	Deletes a JMS resource.
jms-ping	Pings the JMS provider to see if it is running.
list-jmsdest	Lists JMS physical destinations for a server instance.
list-jms-resources	Lists JMS resources for a server instance.
get and set jms-service	Gets and sets attributes for the JMS service.
get and set jms-resource	Gets and sets attributes for a JMS resource.

For information on the syntax of these commands, see the online help for asadmin. For more information on asadmin, as well as a list of attributes for jms-service and jms-resource, see Appendix A, “Using the Command Line Interface.”

Chapter 12

Configuring the Server For CORBA/IIOP Clients

This chapter explains how to configure support for CORBA/IIOP clients, using the RMI/IIOP protocol within the Sun ONE Application Server environment.

This chapter includes the following topics:

- About Support for CORBA/IIOP Clients
- Configuring the ORB

About Support for CORBA/IIOP Clients

The J2EE platform provides indirect support for various types of clients, different hardware platforms, and a multitude of software applications through its interoperabillity requirements. As a J2EE-compliant product, Sun ONE Application Server supports a standard set of protocols and formats that ensure interoperabillity.

The CORBA (Common Object Request Broker Architecture) model is based on clients requesting services from distributed objects or servers through a well-defined interface, by issuing requests to the objects in the form of remote method requests. A remote method request carries information about an operation that needs to be performed including the object name (called an object reference) of the service provider and the actual parameters, if there are any. CORBA automatically handles a lot of network programming tasks such as object registration, object location, object activation, request de-multiplexing, error-handling, marshalling and operation dispatching.

The following topics are covered in this section:

- About Interoperabillity

- About the ORB
- About the RMI/IIOP Functionality
- About the Authentication Process

About Interoperabillity

Interoperabillity essentially means the ability of an enterprise environment to bring together applications written in various languages. One or more of these existing applications may be running on a personal computer platform, while others may be running on UNIX. In addition, these enterprise environments may also be supporting standalone Java technology based applications that are not directly supported by the J2EE platform.

J2EE is mandated to provide support for CORBA IIOP (Internet Inter-Orb Protocol) protocol. CORBA defines a model that specifies interoperabillity between distributed objects on a network in a way that is transparent to the user. CORBA achieves this by defining ways for specifying the externally visible characteristics of a distributed object in a way that is implementation-independent.

About the ORB

Object Request Broker (ORB for short) is the central component of CORBA. The ORB provides the required infrastructure to identify and locate objects, handle connection management, deliver data and request communication.

One CORBA object never talks directly with another. Instead, the object makes requests through a remote stub to the ORB running on the local machine. The local ORB then passes the request to an ORB on the other machine using Internet Inter-Orb Protocol (IIOP for short). The remote ORB then locates the appropriate object (servant) processes the request and returns the results. IIOP can be used as a Remote Method Invocation (RMI for short) protocol by JAVA applications or objects, using the RMI-IIOP technology.

About the RMI/IIOP Functionality

CORBA specifies the ORB which allows applications to communicate with each other regardless of location. This interoperabillity is delivered through IIOP, and is typically found in an Intranet setting. Some of the functionalities achieved by RMI over IIOP are as follows:

- Interoperabillity with objects written in other languages.
- Ability to propagate transaction and security context.
- Plug-and-play environment for ORB services.
- Interoperabillity with EJBs
- Use of the COSNaming service, an IIOP-based naming service. The EJB interoperabillity protocol requires the use of the COSNaming to look up EJB objects using the Java Naming Directory Interface (JNDI for short) API.

The JAVA ORB that comes bundled with Sun ONE Application Server supports the following functionalities:

- Conformance level 0 of CSIv2 (Common Secure Interoperabillity version 2).
- Fully compliant COSNaming service implements the IDL interfaces and aid the EJB container to publish EJBHome references.
- IIOP / GIOP Ver 1.2. CORBA specifies the ORB which allows applications to communicate with each other regardless of location. This interoperabillity is delivered through IIOP.

About the Authentication Process

Authentication is the process of confirming an identity. In the context of network interactions, authentication is the confident identification of one party by another party. Certificates are one way of supporting authentication.

The following two kinds of authentication are applicable:

Server Authentication. Server authentication refers to the confident identification of a server by a client; that is, identification of the organization assumed to be responsible for the server at a particular network address.

Client Authentication. Client authentication refers to the confident identification of a client by a server; that is, identification of the person assumed to be using the client software.

Clients can have multiple certificates, much like a person might have several different pieces of identification.

Configuring the ORB

You can configure multiple IIOP-listeners for each instance of Sun ONE Application Server. By default, one IIOP listener is configured. You can configure the IIOP listener properties for your ORB and add additional listeners.

You can also enable monitoring for the ORB, specify the log level at which messages will be logged, specify thread pool settings, and configure IIOP listener ports and SSL configuration for the IIOP path. In this section, we will discuss how to configure ORB support for an instance of Sun ONE Application Server.

The following topics are included in this section:

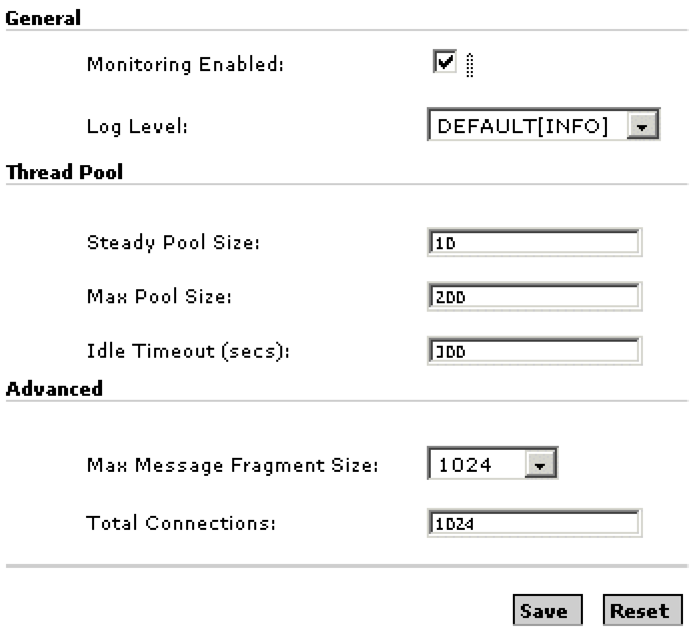
- To Perform General ORB Configuration
- To Configure IIOP Listener For the ORB

To Perform General ORB Configuration

Using the Administration interface, you can enable monitoring, set log levels, and configure pool settings for the thread pool. To perform general ORB configuration, perform the following tasks:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance for which you want to configure ORB settings.
2. Click the ORB tab. You will see the figure "General ORB Configuration" in the right pane of the Administration interface:

Figure 12-1 General ORB Configuration



General

Monitoring Enabled: ☑

Log Level: DEFAULT[INFO]

Thread Pool

Steady Pool Size: 10

Max Pool Size: 200

Idle Timeout (secs): 300

Advanced

Max Message Fragment Size: 1024

Total Connections: 1024

3. In the General section of this window, you can enable monitoring, and set log levels for your ORB.

a. To enable monitoring for the ORB, mark the Monitoring Enabled checkbox.

b. Choose the log level you want, from the Log Level drop-down list. The default log level for the server is typically set to INFO. The default level for the ORB is to use the default for the server. The log level will therefore display Default (INFO), in the drop-down list.

Log levels are provided to record messages of a range of severity, from FINEST to FATAL. Setting a log level allows you to select what granularity of messages are displayed in the log. A granularity of WARNING will display WARNING, ALERT, SEVERE and FATAL messages. Normally you would need to set the granularity at the server-wide level, but you can use this setting to control the messages displayed from the Sun ONE Application Server ORB.

4. In the Thread Pool section of this window, you can specify the pool settings for the request threads used by the ORB.

   Request threads handle user requests for application components. When Sun ONE Application Server receives a request, it assigns the request to a free thread from the thread pool. The thread executes the client's requests and returns results. For example, if the request needs to use a system resource that is currently busy, the thread waits until that resource is free before allowing the request to use that resource.

   You can specify the minimum and maximum number of threads that are reserved for requests from applications. The thread pool is dynamically adjusted between these two values. The minimum thread-pool size you specify signals the ORB to allocate at least that many threads in reserve for application requests. That number is increased upto the maximum thread-pool size that you specify.

   Increasing the number of threads available to a process allows the process to respond to more application requests simultaneously.

   a. In the Steady Pool Size field, specify the minimum number of threads in the pool. The pool will also shrink to this number after threads are idle for the period specified in the Idle Timeout (secs) field.

   b. In the Max Pool Size field, specify the maximum number of threads to which the thread pool can grow.

   c. In the Idle Timeout (secs) field, specify the timeout for the idle threads in the threadpool to be cleaned up.

5. In the Advanced section of this window, you can configure advanced options for your ORB, as follows:

   a. In the Message Fragment Size field, specify the maximum GIOP 1.2 message size, in order to support fragmentation. The default fragment size is 1024.

   b. In the Total Connections field, specify the maximum number of incoming remote IIOP connections allowed by the ORB server process.

6. Click Save to save your settings. If you want to revert to your previous settings without saving the recent changes, click Revert.

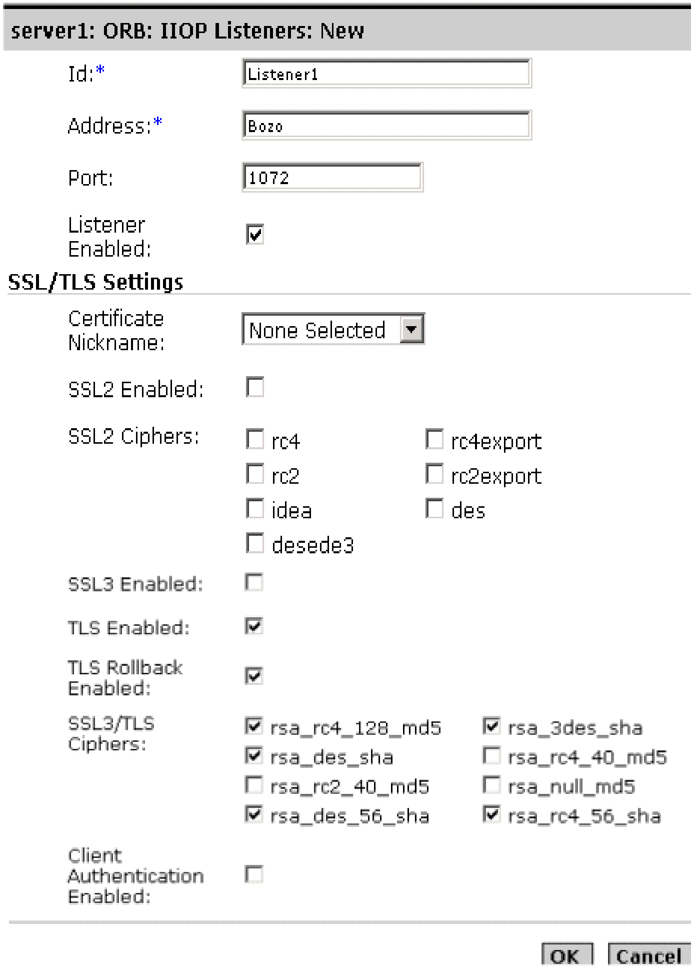
To Configure IIOP Listener For the ORB

Each new instance of Sun ONE Application Server comes with a default ORB configuration, which includes a pre-configured IIOP listener. The IIOP listener is a listen socket that listens on a specified port and accepts incoming connections from CORBA based client application You can configure any number of IIOP listeners for a single instance of Sun ONE Application Server.

To create a new IIOP listener or to configure IIOP listener properties, perform the following tasks:

1. In the left pane of the Administration interface, expand the Sun ONE Application Server instance for which you want configure ORB properties.
2. Click ORB, and open the IIOP Listener tab under it. You will see a list of all the IIOP Listeners that have been configured for that specific instance of Sun ONE Application Server.
3. To create a new IIOP Listener, click New (if you are editing an existing IIOP listener, just open the listener and perform tasks listed in the following steps). When you click New, or when you open an existing IIOP listener, you will see the figure "Creating a New IIOP Listener":

Figure 12-2 Creating a New IIOP Listener



server1: ORB: IIOP Listeners: New

Id:* Listener1

Address:* Bozo

Port: 1072

Listener Enabled: ☑

SSL/TLS Settings

Certificate Nickname: None Selected

SSL2 Enabled: □

SSL2 Ciphers: □ rc4 □ rc4export □ rc2 □ rc2export □ idea □ des □ desede3

SSL3 Enabled: □

TLS Enabled: ☑

TLS Rollback Enabled: ☑

SSL3/TLS Ciphers: ☑ rsa_rc4_128_md5 ☑ rsa_3des_sha ☑ rsa_des_sha □ rsa_rc4_40_md5 □ rsa_rc2_40_md5 □ rsa_null_md5 ☑ rsa_des_56_sha ☑ rsa_rc4_56_sha

Client Authentication Enabled: □

4. You can configure general parameters for your IIOP listener, as follows:

a. In the Id text field, provide a name to identify the listener. You can use any identifier, such as *ORB_Listener1*, *ORB_Listener2*, etc.

b. In the Address text field, type the address of the machine on which you have installed Sun ONE Application Server. You can either specify the machine address in the `machinename.domainname` format, as indicated in the given example, or you can provide the IP address of the machine.

c. In the Port text field, type a unique port number for the new IIOP Listener. The default IIOP listener comes with a default port number. You can change this port number. However, before changing the port number, please ensure that the new port number that you specify is not being used by any other existing software application or process.

d. To enable the listener, mark the Listener Enabled checkbox.

5. In the SSL/TLS Settings section on this page, you can set security for the IIOP listener. Check the appropriate boxes associated with the Secure Sockets Layer (SSL) and Transport Layer Security (TLS), including all the ciphers. You can select either SSL2 or SSL3/TLS sockets.You can configure the SSL/TLS settings for your listener, as follows:

   a. In the Certificate Nickname field, provide the nickname of the certificate that the server presents to the client during SSL handshake. You must have previously installed a certificate to see its nickname in this list.

   b. Mark the SSL2 Enabled field, to enable SSL2 security option for the listener path.

   c. Select the SSL2 ciphers that you want to use for the SSL2 security. Mark the checkboxes against the required ciphers. Unless you have a compelling reason for not using a specific cipher suite, you should allow them all.

   d. Mark the SSL3 Enabled field, to enable SSL3 security option for the listener path.

   e. Mark the TLS Enabled field, to enable TLS. TLS must also be enabled on the browser seeking access to your server. Check both TLS and SSL3 for Netscape Navigator 6.0.

   f. Mark the TLS Rollback Enabled field. In order to enable TLS Rollback, you need to enable TLS first. Also ensure that SSL3 and SSL2 are disabled, when you enable this option. Use the TLS Rollback option for Microsoft Internet Explorer 5.0 and 5.5.

   g. Select the SSL3/TLS ciphers that you want to use for SSL3 and TLS. Select these only if you have enabled SSL3 or TLS. Unless you have a compelling reason for not using a specific cipher suite, you should allow them all.

   h. Mark the Client Authentication Enabled checkbox to indicate whether the ORB listener port for SSL IIOP connections with client authentication is enabled or not. Client authentication is the process of authenticating client certificates by cryptographically verifying the certificate signature and the certificate chain leading to the CA on the trust CA list.

6. Click OK to save the IIOP listener settings.

NOTE

- When you install Sun ONE Application Server, an IIOP listener is created for the default server instance. The default port number for the default IIOP listener port is 3700.
- Please note that each IIOP listener must bear a different port number. Also note that the machine address that you provide in the Address text field must be the address of the machine on which Sun ONE Application Server is installed.
- For more information about SSL settings for the listener path, and other details of security for Sun ONE Application Server, see the *Sun ONE Application Server Administrator's Guide to Security*.

Chapter 13

Deploying Applications

This chapter describes how to deploy various Sun ONE Application Server modules and applications.

Sun ONE Application Server modules and applications include J2EE standard elements and Sun ONE Application Server specific elements. Only Sun ONE Application Server specific elements are described in detail in this chapter.

To know about packaging and assembling modules and applications for deployment, see the *Sun ONE Application Server Developer's Guide*.

This chapter includes the following topics:

- About J2EE Modules
- About J2EE Applications
- J2EE Standard Descriptors
- Sun ONE Application Server Descriptors
- Naming Standards
- Deployment Directory Structure
- Runtime Environments
- About Classloaders
- Deploying Modules and Applications
- The Application Deployment Descriptor Files

About J2EE Modules

A J2EE module is a collection of one or more J2EE components of the same container type with deployment descriptors of that type. One descriptor is J2EE standard, the other is Sun ONE Application Server specific. Types of J2EE modules are as follows:

- **Web Application Archive (WAR):** A web application is a collection of servlets, HTML pages, classes, and other resources that can be bundled and deployed to several J2EE application servers. A WAR file can consist of the following items: servlets, JSPs, JSP tag libraries, utility classes, static pages, client-side applets, beans, bean classes, and deployment descriptors (`web.xml` and optionally `sun-web.xml`).
- **EJB JAR File:** The EJB JAR file is the standard format for assembling enterprise beans. This file contains the bean classes (home, remote, local, and implementation), all of the utility classes, and the deployment descriptors (`ejb-jar.xml` and optionally `sun-ejb-jar.xml`). If the EJB is an entity bean with container managed persistence, a CMP deployment descriptor, `sun-cmp-mapping.xml`, may be included as well.
- **Application (RMI/IIOP) Client JAR File:** An RMI/IIOP Client is a Sun ONE Application Server specific type of J2EE client. An RMI/IIOP Client supports the standard J2EE Application Client specifications, and in addition, supports direct access to the Sun ONE Application Server. Its deployment descriptors are `application-client.xml` and optionally `sun-application-client.xml`.
- **Resource RAR File:** RAR files apply to J2EE CA connectors. A connector module is like a device driver. It is a portable way of allowing EJBs to access a foreign enterprise system. Each Sun ONE Application Server connector has a J2EE XML file, `ra.xml`. A connector must also have a Sun ONE Application Server deployment descriptor, `sun-ra.xml`.

Package definitions must be used in the source code of all modules so the classloader can properly locate the classes after the modules have been deployed.

Because the information in a deployment descriptor is declarative, it can be changed without requiring modifications to source code. At run time, the J2EE server reads this information and acts accordingly.

EJB JAR and Web modules can also be assembled as separate `.jar` or `.war` files and deployed separately, outside of any application, as in the following figure.

About J2EE Applications

A J2EE application is a logical collection of one or more J2EE modules tied together by application deployment descriptors. Components can be assembled at either the module or the application level. Components can also be deployed at either the module or the application level.

Components are assembled into modules and then assembled into a Sun ONE Application Server application `.ear` file ready for deployment.

Each module has a Sun ONE Application Server deployment descriptor and a J2EE deployment descriptor. The Sun ONE Application Server Administration interface uses the deployment descriptors to deploy the application components and to register the resources with the Sun ONE Application Server.

An application consists of one or more modules, an optional Sun ONE Application Server deployment descriptor, and a required J2EE application deployment descriptor. All items are assembled, using the Java ARchive (`.jar`) file format, into one file with an extension of `.ear`.

J2EE Standard Descriptors

The J2EE platform provides assembly and deployment facilities. These facilities use JAR files as the standard package for components and applications, and XML-based deployment descriptors for customizing parameters. For more information on the J2EE assembly and deployment process, see *Developing Enterprise Applications with the J2EE*, v 1.0, Chapter 7.

The J2EE standard deployment descriptors are described in the J2EE specification, v1.3.

To check the correctness of these deployment descriptors prior to deployment, see the information on the deployment descriptor verifier in the *Sun ONE Application Server Developer's Guide*.

The following table, "J2EE Standard Descriptors," shows where to find more information about J2EE standard deployment descriptors. The left column lists the deployment descriptors, and the right column lists where to find more information about those descriptors.

Table 13-1 J2EE Standard Descriptors

Deployment Descriptor	Where to Find More Information
`application.xml`	Java 2 Platform Enterprise Edition Specification, v1.3, Chapter 8, "Application Assembly and Deployment - J2EE:application XML DTD"
`web.xml`	Java Servlet Specification, v2.3 Chapter 13, "Deployment Descriptor," and JavaServer Pages Specification, v1.2, Chapter 7, "JSP Pages as XML Documents," and Chapter 5, "Tag Extensions"
`ejb-jar.xml`	Enterprise JavaBeans Specification, v2.0, Chapter 16, "Deployment Descriptor"
`application-client.xml`	Java 2 Platform Enterprise Edition Specification, v1.3, Chapter 9, "Application Clients - J2EE:application-client XML DTD"
`ra.xml`	Java 2 Enterprise Edition, J2EE Connector Architecture Specification, v1.0, Chapter 10, "Packaging and Deployment."

You can find specifications here:

`http://java.sun.com/products/`

Sun ONE Application Server Descriptors

Sun ONE Application Server uses additional deployment descriptors for configuring features specific to the Sun ONE Application Server. These are optional except for the `sun-ra.xml` file, which is required for a connector module.

To check the correctness of these deployment descriptors prior to deployment, see the information on the deployment descriptor verifier in the *Sun ONE Application Server Developer's Guide*.

The following table, "Sun ONE Application Server Descriptors," shows where to find more information about Sun ONE Application Server deployment descriptors. The left column lists the deployment descriptors, and the right column lists where to find more information about those descriptors.

Table 13-2 Sun ONE Application Server Descriptors

Deployment Descriptor	Where to Find More Information
`sun-application.xml`	"The Application Deployment Descriptor Files" on page 358.
`sun-web.xml`	*Sun ONE Application Server Developer's Guide to Web Applications*
`sun-ejb-jar.xml` and `sun-cmp-mapping.xml`	*Sun ONE Application Server Developer's Guide to Enterprise Java Beans*
`sun-application-client.xml` and `sun-acc.xml`	*Sun ONE Application Server Developer's Guide to Clients*
`sun-ra.xml`	*Sun ONE J2EE CA Service Provider Implementation Administrator's Guide*

NOTE The Sun ONE Application Server deployment descriptors must have 600 level access privileges on UNIX systems.

The DTD schema files for all the Sun ONE Application Server deployment descriptors are located in the *install_dir*/`appserv/lib/dtds` directory.

Naming Standards

Names of applications and individually deployed EJB JAR, WAR, and connector RAR modules (as specified by the `name` attributes in the `server.xml` file) must be unique in the Sun ONE Application Server. If you do not explicitly specify a name, the default name is the first portion of the file name (without the `.war` or `.jar` extension). For details about `server.xml`, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Modules of different types can have the same name within an application, because when the application is deployed, the directories holding the individual modules are named with `_jar`, `_war` and `_rar` suffixes. Modules of the same type within an application must have unique names. In addition, database schema file names must be unique within an application.

Using a Java package-like naming scheme is recommended for module filenames, EAR filenames, module names as found in the `<module-name>` portion of the `ejb-jar.xml` files, and EJB names as found in the `<ejb-name>` portion of the `ejb-jar.xml` files. The use of this package-like naming scheme ensures that name collisions do not occur. The benefits of this naming practice apply not only to the Sun ONE Application Server, but to other J2EE application servers as well.

JNDI lookup names for EJBs must also be unique. Here too, establishing a consistent naming convention may help. For example, appending the application name and the module name to the EJB name would be one way to guarantee unique names. In this case, `mycompany.pkging.pkgingEJB.MyEJB` would be the JNDI name for an EJB in the module `pkgingEJB.jar`, which is packaged in the application `pkging.ear`.

Make sure your package and file names do not contain spaces or characters that are illegal for your operating system.

Deployment Directory Structure

When you deploy an application, the directories holding the individual modules are named with `_jar`, `_war` and `_rar` suffixes. If you use the `asadmin deploydir` command to deploy a directory instead of an EAR file, your directory structure must follow this convention.

Module and application directory structures follow the structure outlined in the J2EE specification.

Here is an example directory structure of a simple application containing a web module, an EJB module, and a client 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
```

Here is an example directory structure of an individually deployed connector module.

```
+ MyConnector/
|--- readme.html
|--- ra.jar
|--- client.jar
|--- win.dll
|--- solaris.so
'--+ META-INF/
   |--- MANIFEST.MF
   |--- ra.xml
   '--- sun-ra.xml
```

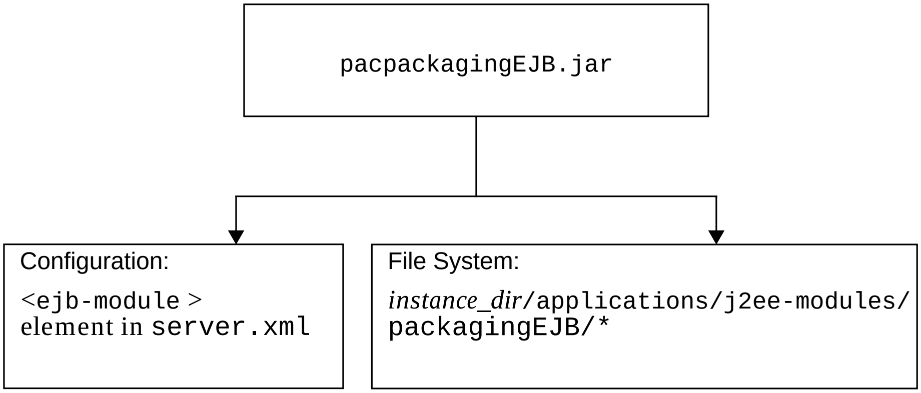
Runtime Environments

Whether you deploy a component as an individually deployed module or as an application, deployment affects both the file system and the server configuration. See the "Module Runtime Environment" and "Application Runtime Environment" figures.

Module Runtime Environment

The following figure,"Module Runtime Environment," illustrates the environment for individually deployed module-based deployment.

Figure 13-1 Module Runtime Environment


pacpackagingEJB.jar
Configuration:
<ejb-module > element in server.xml
File System:
instance_dir/applications/j2ee-modules/packagingEJB/*

For file system entries, modules are extracted as follows:

instance_dir/`applications/j2ee-modules/`*module_name*
instance_dir/`generated/ejb/j2ee-modules/`*module_name*
instance_dir/`generated/jsp/j2ee-modules/`*module_name*

The `generated/ejb` directory contains stubs and ties; the `generated/jsp` directory contains compiled JSPs.

Lifecycle modules are extracted as follows:

instance_dir/`applications/lifecycle-modules/`*module_name*

Configuration entries are added in `server.xml` as follows:

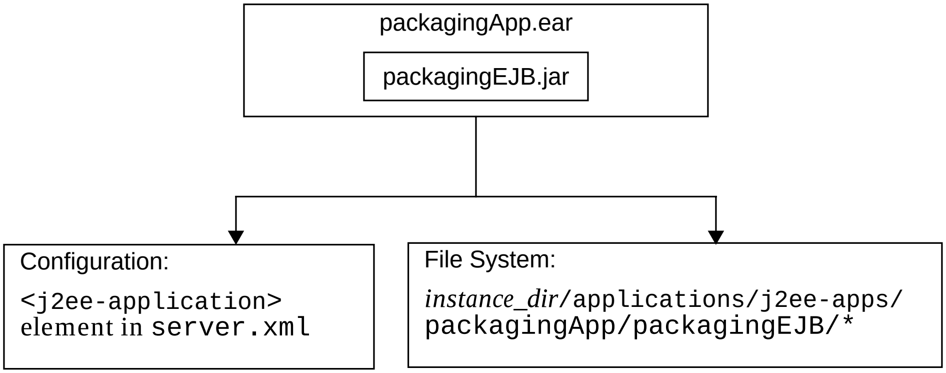
```
<server>
   <applications>
      <type-module>
         ...module configuration...
      </type-module>
   </applications>
</server>
```

The *type* of the module in `server.xml` can be `lifecycle`, `ejb`, `web`, or `connector`. For details about `server.xml`, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Application Runtime Environment

The following figure, "Application Runtime Environment," illustrates the environment for application-based deployment.

Figure 13-2 Application Runtime Environment


packagingApp.ear
packagingEJB.jar
Configuration:
<j2ee-application> element in server.xml
File System:
instance_dir/applications/j2ee-apps/packagingApp/packagingEJB/*

For file system entries, applications are extracted as follows:

instance_dir/`applications/j2ee-apps/`*app_name*
instance_dir/`generated/ejb/j2ee-apps/`*app_name*
instance_dir/`generated/jsp/j2ee-apps/`*app_name*

The `generated/ejb` directory contains stubs and ties; the `generated/jsp` directory contains compiled JSPs.

Configuration entries are added in `server.xml` as follows:

```
<server>
   <applications>
      <j2ee-application>
         ...application configuration...
      </j2ee-application>
   </applications>
</server>
```

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

Configuring server.xml To Use FastJavac Compiler

By default, Sun ONE Application Server uses the in built JDK compiler to compile applications during deployment. You can also use Sun One Studio's FastJavac compiler, which has a faster compilation rate, during deployment.

In the bundled Solaris install, the location of the FastJavac compiler is not transparent. In order to use the FastJavac compiler, you need to configure the administrative server's server.xml with the path for the compiler, as follows:

Add the following `jvm-option` in the `java-config` element, in server.xml:

```
<java-config java-home="/<install-dir>/jdk" server-classpath="....." >

jvm-options>-Dcom.sun.aas.deployment.java.compiler=/<install-dir>/s
tudio4/bin/fastjavac/fastjavac.sun</jvm-options>

<property name="com.sun.aas.deployment.java.compiler.options"

value="-jdk /<install-dir>/jdk" />

</java-config>
```

About Classloaders

Understanding Sun ONE Application Server classloaders can help you determine where and how you can position supporting JAR and resource files for your modules and applications.

In a Java Virtual Machine (JVM), the classloaders dynamically load a specific java class file needed for resolving a dependency. For example, when an instance of `java.util.Enumeration` needs to be created, one of the classloaders loads the relevant class into the environment. For a more detailed discussion on Classloaders, see the *Sun ONE Application Server Developer's Guide.*

Deploying Modules and Applications

This section describes the different ways to deploy J2EE applications and modules to the Sun ONE Application Server. It covers the following topics:

- Deployment Names and Errors
- The Deployment Life Cycle
- Deployment of Module or Application
- Deploying a WAR Module
- Deploying an EJB JAR Module
- Deploying a Lifecycle Module
- Deploying an RMI/IIOP Client
- Deploying a J2EE CA Resource Adapter
- Deploying Static Content
- Access to Shared Frameworks

Deployment Names and Errors

A unique name is generated in the `server.xml` file when you deploy an application or module. Do not change this name. During deployment, the server detects any name collisions and does not load an application or module having a non-unique name. Messages are sent to the server log when this happens. For more about naming, see "Naming Standards" on page 343.

If an error occurs during deployment, the application or module is not deployed. If a module within an application contains an error, the entire application is not deployed.

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

The Deployment Life Cycle

After an application is initially deployed, it may be modified and reloaded, redeployed, disabled, reenabled, and finally undeployed (removed from the server). This section covers the following topics related to the deployment life cycle:

- Dynamic Deployment
- Disabling a Deployed Application or Module
- Dynamic Reloading

Dynamic Deployment

You can deploy, redeploy, and undeploy an application or module without restarting the server. This is called dynamic deployment.

Although primarily for developers, dynamic deployment can be used in operational environments to bring new applications and modules online without requiring a server restart. Whenever a redeployment is done, the sessions at that transit time become invalid. The client must restart the session.

Disabling a Deployed Application or Module

You can disable a deployed application or module without removing it from the server. Each application and module has an `enabled` attribute in the `server.xml` file and a corresponding option in the Administration interface, which you can change. For details about `server.xml`, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Dynamic Reloading

If dynamic reloading is enabled, you do not have to redeploy an application or module when you change its code. All you have to do is copy the changed class files into the deployment directory for the application or module. The server checks for changes periodically and redeploys the application, automatically and dynamically, with the changes.

This is useful in a development environment, because it allows code changes to be tested quickly. Dynamic reloading is not recommended for a production environment, however, because it may degrade performance. In addition, whenever a reload is done, the sessions at that transit time become invalid. The client must restart the session.

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.

- Edit the following attributes of the `server.xml` file's `applications` element:
  - `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 ONE 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 `reload-interval="-1"`.

Tools for Deployment

This section discusses the various tools that can be used to deploy modules and applications. The deployment tools include:

- The asadmin Utility
- The Administration Interface
- Sun ONE Studio

The asadmin Utility

You can use the asadmin utility to deploy or undeploy applications and individually deployed modules on local servers. Concurrent deployment on multiple machines is not supported. This section describes the asadmin utility only briefly.

To deploy a lifecycle module, see "Deploying a Lifecycle Module" on page 355.

asadmin deploy

The asadmin deploy command deploys a WAR, JAR, RAR, or EAR file. To deploy an application, specify --type application in the command. To deploy an individual module, specify --type ejb, web, connector. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin deploy --user admin_user [--password admin_password] [--host
localhost] [-port 4848] [--secure | -s] [--virtualservers
virtual_servers] [--type application|ejb|web|connector] [--contextroot
contextroot] [--force=true] [--precompilejsp=false] [--name
component_name] [--upload=true] [--retrieve local_dirpath] [--instance
instance_name] filepath
```

For example, the following command deploys an individual EJB module:

```
asadmin deploy --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB.jar
```

asadmin deploydir

The asadmin deploydir command deploys an application or module in an open directory structure. The structure must be as specified in "Deployment Directory Structure" on page 344. The location of the *dirpath* under *instance_dir*/applications/j2ee-apps or *instance_dir*/applications/j2ee-modules determines whether it is an application or individually deployed module. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin deploydir --user admin_user [--password admin_password] [--host
localhost] [-port 4848] [--secure | -s] [--virtualservers
virtual_servers] [--type application|ejb|web|connector] [--contextroot
contextroot] [--force=true] [--precompilejsp=false] [--name
component_name] [--instance instance_name] dirpath
```

For example, the following command deploys an individual EJB module:

```
asadmin deploydir --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB
```

asadmin undeploy

The `asadmin undeploy` command undeploys an application or module. To undeploy an application, specify `--type app` in the command. To undeploy a module, specify `--type ejb`, `web`, or `connector`. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin undeploy --user admin_user [--password admin_password] [--host
localhost] [-port 4848] [--secure | -s] [--type
application|ejb|web|connector] [--instance instance_name] component_name
```

For example, the following command undeploys an individual EJB module:

```
asadmin undeploy --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB
```

The Administration Interface

You can use the Administration interface to deploy modules and applications to both local and remote Sun ONE Application Server sites. To use this tool, follow these steps:

1. Open the Applications component under your server instance.

2. Go to the Enterprise Applications, Web Applications, Connector Modules, or EJB Modules page.

3. Click on the Deploy button.

4. Enter the full path to the module or application (or click on Browse to find it), then click on the OK button.

5. Enter the module or application name.

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

6. Assign the application or module to one or more virtual servers by checking the boxes next to the virtual server names.

7. Click on the OK button.

To deploy a lifecycle module, see "Deploying a Lifecycle Module" on page 355.

Sun ONE Studio

You can use Sun ONE Studio 4 to deploy J2EE applications and modules. For more information about using Sun ONE Studio, see the *Sun ONE Studio 4, Enterprise Edition Tutorial*.

NOTE In Sun ONE Studio, deploying a module or application is referred to as *executing* it. Execution also includes making sure the server is running and displaying the correct URL to activate the module or application.

Deployment of Module or Application

You can deploy applications or individual modules that are independent of applications. The runtime and file system implications of application-based or individual module-based deployment are described in "Runtime Environments" on page 346.

Individual module-based deployment is preferable when components need to be accessed by:

- Other modules
- J2EE Applications
- RMI/IIOP clients (Module-based deployment allows shared access to a bean from an RMI/IIOP client, a servlet, or an EJB.)

Modules can be combined into an EAR file and then deployed as a single module. This is similar to deploying the modules of the EAR independently.

Deploying a WAR Module

You deploy a WAR module in one of the ways described in "Tools for Deployment" on page 351.

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 WAR module, 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. For more information about the `-keepgenerated` property, see the *Sun ONE Application Server Developer's Guide to Web Applications*.

Deploying an EJB JAR Module

You deploy an EJB JAR module in one of the ways described in "Tools for Deployment" on page 351.

You can keep the generated source for stubs and ties by adding the `-keepgenerated` flag to the `rmic-options` attribute of the `java-config` element in `server.xml`. If you include this flag when you deploy the EJB JAR module, the generated source is kept in *instance_dir*`/generated/ejb/j2ee-apps/`*app_name*`/`*module_name* if it is in an application or *instance_dir*`/generated/ejb/j2ee-modules/`*module_name* if it is in an individually deployed EJB JAR module. For more information about the `-keepgenerated` flag, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Deploying a Lifecycle Module

For general information about lifecycle modules, see the *Sun ONE Application Server Developer's Guide*.

You can deploy a lifecycle module using the following tools:

- The asadmin Utility
- The Administration Interface

The asadmin Utility

To deploy a lifecycle module, use the `asadmin create-lifecycle-module` command. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin create-lifecycle-module --user admin_user [--password
admin_password] [--host localhost] [--port 4848] [--secure | -s]
[--instance instance_name] --classname classname [--classpath classpath]
[--loadorder load_order_number] [--failurefatal=false] [--enabled=true]
[--description text_description] [--property (name=value)[:name=value]*]
modulename
```

For example:

```
asadmin create-lifecycle-module --user jadams --password secret
--host localhost --port 4848 --instance server1 --classname
RMIServer MyRMIServer
```

To undeploy a lifecycle module, use the `asadmin delete-lifecycle-module` command. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin delete-lifecycle-module --user admin_user [--password
admin_password] [--host localhost] [-port 4848] [--secure | -s]
[--instance instance_name] module_name
```

For example:

```
asadmin delete-lifecycle-module --user jadams --password secret
--host localhost --port 4848 --instance server1 MyRMIServer
```

To list the lifecycle modules that are deployed on a server instance, use the `asadmin list-lifecycle-modules` command. The syntax is as follows, with defaults shown for optional parameters that have them:

```
asadmin list-lifecycle-modules --user admin_user [--password
admin_password] [--host localhost] [-port 4848] instance_name
```

For example:

```
asadmin list-lifecycle-module --user jadams --password secret --host
localhost --port 4848 server1
```

The Administration Interface

You can also use the Administration interface to deploy a lifecycle module. Follow these steps:

1. Open the Applications component under your server instance.
2. Go to the Life Cycle Modules page.
3. Click on the Deploy button.
4. Enter the following information:
   - Name (required) - The name of the life cycle module.
   - Class Name (required) - The fully qualified name of the life cycle module's class file.
   - Classpath (optional) - The classpath for the life cycle module. Specifies where the module is located. The default location is under the application root directory.
   - Load Order (optional) - Determines the order in which life cycle modules are loaded at startup. Modules with smaller integer values are loaded sooner. Values can range from `101` to the operating system's `MAXINT`. Values from `1` to `100` are reserved.

- Failure Fatal (optional) - Determines whether the server is shut down if the life cycle module fails. The default is `false`.
- Enable (optional) - Determines whether the life cycle module is enabled. The default is `true`.

5. Click on the OK button.

Deploying an RMI/IIOP Client

Deployment is only necessary for clients that communicate with EJBs. Deploying an RMI/IIOP client is a three-step process:

1. Deploy the EAR or EJB JAR to be accessed by the RMI/IIOP client.
2. Assemble the necessary client files and deploy the client.
3. After deployment, a client JAR file is created in the following location for an application:

   instance_dir`/applications/j2ee-apps/`*app_name*/*app_name*`Client.jar`

   or in the following location for an individually deployed module:

   instance_dir`/applications/j2ee-modules/`*module_name*/*module_name*`Client.jar`

   The client JAR contains the ties and necessary classes for the RMI/IIOP client. Copy this file to the client machine, and set the APPCPATH environment variable on the client to point to this JAR.

You are now ready to run the client. For more information, see the *Sun ONE Application Server Developer's Guide to Clients*.

Deploying a J2EE CA Resource Adapter

You deploy a connector module in one of the ways described in "Tools for Deployment" on page 351.

Deploying Static Content

Static content (HTML, images, etc.) can be hosted both on the web server and on the Sun ONE Application Server. However, when a WAR is registered, the static content gets deployed on the application server. All of the samples shipped with Sun ONE Application Server host the static content on the application server.

For example, to access a static file `index.html` on the application server, use:

`http://`*server*`:`*port*`/NASApp/&lt`*context_root*`/index.html`

Access to Shared Frameworks

When J2EE applications and modules use shared framework classes (such as components and libraries) the classes can be put in the path for the System Classloader or the Common Classloader rather than in an application or module. If you assemble a large, shared library into every module that uses it, the result is a huge file that takes too long to register with the server. In addition, several versions of the same class could exist in different classloaders, which is a waste of resources.

For more information about the system classloader, see "About Classloaders" on page 348.

The Application Deployment Descriptor Files

Sun ONE Application Server applications include two deployment descriptor files:

- A J2EE standard file (`application.xml`), described in the Java Servlet Specification, v2.3, Chapter 13, "Deployment Descriptors."
- An optional Sun ONE Application Server specific file (`sun-application.xml`), described in this section.

For more information on application deployment descriptor files, see the *Sun ONE Application Server Developer's Guide*.

Part 3

Managing HTTP Server Features and Virtual Servers

Chapter 14

Configuring HTTP Features

This chapter describes how to configure preferences for your HTTP-related features of your Sun ONE Application Server. For preferences related to virtual servers and HTTP listeners, see Chapter 15, “Using Virtual Servers.”

This chapter includes the following topics:

- About the HTTP Features
- Configuring the File Cache
- Tuning Your Server for Performance
- Configuring HTTP Quality of Service
- Adding and Using Thread Pools
- Configuring the File Cache
- Editing Advanced Settings
- Configuring MIME Types

About the HTTP Features

The Sun ONE Application Server HTTP features include setting performance levels for application server instances, setting performance tuning-related parameters, and using the file cache to improve performance. These settings are stored in two configuration files: `init.conf` and `server.xml`. You edit the `init.conf` settings on the Advanced Settings page. For more information, see “Editing Advanced Settings” on page 366.

Other properties you edit are stored in the `server.xml` file, in the `http-service` element. For more information on both the `init.conf` file and the `server.xml` file, see the *Sun ONE Application Server Administrator’s Configuration File Reference*.

Configuring the File Cache

The Sun ONE Application Server uses a file cache to serve static information faster. The file cache contains information about files and static file content. The file cache also caches information that is used to speed up processing of server-parsed HTML.

The file cache is turned on by default. The file cache settings are contained in a file called `nsfc.conf`. This file is present only if file cache parameters have been changed from their defaults. For more information on `nsfc.conf`, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

To configure the file cache:

1. In the left pane, click HTTP Server
2. Click the File Caching tab
3. Enter the desired value in the fields.
4. Click OK.

For more information on using the file cache to improve performance, see the *Sun ONE Application Server Performance Tuning and Sizing Guide*

Tuning Your Server for Performance

On the Performance Tuning page you can configure settings that control your Sun ONE Application Server's performance by controlling how many requests it can handle, how long requests remain open without activity before timing out, and whether you are doing reverse lookups of a client's IP using DNS. Also, if you are using DNS, you can set such performance-related features as whether you are using asynchronous DNS, and DNS caching settings.

For more information on tuning, see the *Sun ONE Application Server Performance Tuning Guide*.

To set the performance tuning settings:

1. In the left pane, click HTTP Server
2. Click the Tuning tab
3. Enter the desired value in the fields.
4. Click OK.

For additional information on the settings you can tune through the Administration interface, see the online help.

Configuring HTTP Quality of Service

Quality of service refers to the performance limits you set for a server. For example, an ISP might want to charge different amounts of money for virtual servers depending on how much bandwidth allowed them.

Before you can use quality of service for a specific virtual server, you must enable it for the server instance and set some values.

To configure the quality of service settings for the server instance:

1. In the left pane, click HTTP Server.
2. Click the QOS tab.
3. To enable quality of service as a whole, click Enable.

   By default quality of service is disabled. Enabling quality of service increases server overhead slightly.

4. Choose the Recompute Interval.

   The recompute interval is the number of milliseconds between each computation of the bandwidth. The default is 100 milliseconds.

5. Choose the Metric Interval.

   The metric interval is the interval in seconds during which the traffic is measured. The default is 30 seconds. All bandwidth measured during this time is averaged to give the bytes per second.

   If your site has a lot of large file transfers, use a large value (several minutes or more) or this field. A large file transfer might take up all the allowed bandwidth for a short metric interval, and result in connections being denied if you've enforced the maximum bandwidth setting. Since the bandwidth is averaged by the metric interval, a longer interval smooths out spikes caused by large files.

   If the bandwidth limit is much lower than available bandwidth (for example, 1 MB-per-second bandwidth limit but with a 1 GB-per-second connection to the backbone), the metric interval should be shortened.

   Please note that if you have large static file transfers and a bandwidth limit that is much lower than available bandwidth, you have to decide which situation to tune for, since the problems require opposite solutions.

6. Set the bandwidth limit, in bytes per second, for the server.

7. Choose whether or not to enforce the bandwidth limit setting.

   If you choose to enforce the bandwidth limit, once the server reaches its bandwidth limit additional connections are refused.

   If you do not enforce the bandwidth limit, when the limit is exceeded the server logs a message to the error log.

8. Choose the maximum number of connections allowed for the server.

   This number is the number of concurrent requests processed.

9. Choose whether or not to enforce the connection limit setting.

   If you choose to enforce the connection limit, once the server reaches its limit additional connections are refused.

   If you do not enforce the connection limit, when the limit is exceeded the server logs a message to the error log.

10. To specify additional name/value pairs, click the Properties button.

11. Click OK.

To configure quality of service using the command-line interface's `asadmin` utility, use the following commands:

- `create-http-qos`
- `delete-http-qos`

These commands use the following syntax:

```
asadmin create-http-qos --user admin_user [--password password] [--host
hostname] [--port admin_port] [--secure | -s] [--passwordfile file_name]
[--virtualserver virtual_server_id] [--bwlimit bandwidth_limit]
[--enforcebwlimit enforce_bandwidth_limit] [--connlimit connection_limit]
[--enforceconnlimit enforce_connection_limit] instancename
```

```
asadmin delete-http-qos --user admin_user [--password password] [--host
hostname] [--port admin_port] [--secure | -s] [--passwordfile
file_name][--virtualserver virtual_server_id] instancename
```

If you specify a virtual server, these commands create or delete quality of service information for that virtual server. If you do not specify a virtual server, the command affects the server instance.

For more information on command syntax, see the command-line interface help. For more informtion on using `asadmin`, see Appendix A, "Using the Command Line Interface."

For more information on the limitations to the quality of service features, see "Administering the Transaction Service Using the CLI" on page 153.

Adding and Using Thread Pools

You can use thread pools to allocate a certain number of threads to a specific service, so that it doesn't take up more threads than you want it to. Another use for thread pools is for running thread-unsafe plugins. By defining a pool with the maximum number of threads set to 1, only one request is allowed into the specified service function.

When you add a thread pool, the information you specify includes the minimum and maximum number of threads, the stack size, and the queue size.

To add a thread pool:

1. In the left pane, click HTTP Server
2. Click Thread Pool.
3. Enter the desired value in the fields.
4. Click OK.

The thread pool is displayed at the bottom of the page. To edit or delete a thread pool, click the Edit or Delete button next to it.

After you've set up a thread pool, use it by designating it as the thread pool for a specific service.

For more information on using thread pools to improve performance, see the *Performance Tuning and Sizing Guide*.

Editing Advanced Settings

When the Sun ONE Application Server starts up, it looks in a file called `init.conf` in the *instance_dir*`/config/` directory to establish a set of global variable settings that affect the server's behavior and configuration. Sun ONE Application Server executes all the directives defined in `init.conf`.

These settings are shown on the Advanced Settings page. You can edit certain settings in the `init.conf` file that affect the following areas:

- DNS
- SSL
- Performance
- CGI
- Keep-alive
- Logging

For a complete description of the `init.conf` file, see the *Sun ONE Application Server Administrator's Configuration File Reference.*

To edit the advanced settings:

1. In the left pane, click HTTP Server
2. Click the Advanced tab
3. Click the type of settings you want to change (DNS, SSL, and so forth).
4. Make the desired changes to the settings and click OK.

For more information about each type of settings, see the online help.

Configuring MIME Types

The Mime Types page allows you to edit your server's MIME files. MIME (Multi-purpose Internet Mail Extension) types control what types of multimedia files your system supports. MIME types also specify what file extensions belong to certain server file types, for example to designate what files are CGI programs.

You can create as many MIME types files as you need, and associate them with the application server instance or virtual servers. One MIME types file, `mime.types`, exists by default on the server, and cannot be deleted.

To create a new MIME Types file:

1. In the left pane, under HTTP Server, click MIME Type File.
2. On the right pane, click New.
3. Enter an identifier for the MIME file, and a file name.
4. Click OK.

To edit the definitions in a MIME file:

1. In the left pane, under HTTP Server, click the icon next to MIME Type File to expand the view.
2. Click the ID of the MIME file you want to edit.
3. On this page, edit the MIME file name associated with the ID.
4. To edit the file extensions in the MIME file, click Edit MIME file.
5. To edit an existing entry, click Edit next to it.
6. On the page that comes up, make your changes and click Change MIME Type.
7. To delete a MIME type, click Remove next to it.
8. To add a new MIME type, enter a category, content type, and file suffix in the fields, then click New Type.

To configure MIME types using the command-line interface's asadmin utility, use the following commands:

- `create-mime`
- `delete-mime`
- `list-mimes`

These commands use the following syntax:

```
asadmin create-mime --user admin_user [--password password] [--host
hostname] [--port admin_port] [--secure | -s] [--passwordfile file_name]
[--instance instancename] --mimefile filename mime_id
```

```
asadmin delete-mime --user admin_user [--password password] [--host
hostname] [--port admin_port] [--secure | -s] [--passwordfile file_name]
[--instance instancename] mime_id
```

```
asadmin list-mimes --user admin_user [--password password] [--host
hostname] [--port admin_port] [--secure | -s] [--passwordfile file_name]
instancename
```

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, “Using the Command Line Interface.”

For information on using the MIME types with virtual servers, see the online help and Chapter 15, “Using Virtual Servers.”

Chapter 15

Using Virtual Servers

This chapter explains how to set up and administer virtual servers using your Sun ONE Application Server. For information on configuring settings for virtual server content, see Chapter 16, "Managing Virtual Server Content."

This chapter includes the following topics:

- Virtual Servers Overview
- Using Sun ONE Application Server Features with Virtual Servers
- Creating and Configuring HTTP Listeners
- Creating and Configuring Virtual Servers
- Deploying Virtual Servers

Virtual Servers Overview

When you use virtual servers you can offer companies or individuals domain names, IP addresses, and some server monitoring capabilities with a single installed server. For the users, it is almost as if they have their own web servers, though you provide the hardware and maintain the virtual servers.

When you install the unbundled version of the Sun ONE Application Server, a default virtual server for the application server instance is created. That is, for the default application server instance `server1`, a virtual server named `server1` is also created. If you are using the Solaris 9 bundled version, you need to create a server instance. When you create it, a virtual server with the same name is also created. A virtual server is created for each additional application server instance you create. For more information on creating and configuring virtual servers, see "Creating and Configuring Virtual Servers" on page 380. For more information on deploying virtual servers, see "Deploying Virtual Servers."

This virtual server controls the Sun ONE Application Server's HTTP features that are available on a per-virtual server basis. You may not want to use multiple virtual servers, but you still configure certain properties for your application server instance by configuring the default virtual server created with that application server instance.

The settings for virtual servers are stored in the `virtual-server` element in the `server.xml` file, found in the *instance_dir*/`config` directory. For more information about this file, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Some information pertaining to a virtual server is stored in its `obj.conf` file. Each virtual server has a separate `obj.conf` file.

This section includes the following topics:

- HTTP Listeners
- Virtual Servers
- The obj.conf File
- Virtual Server Selection for Request Processing
- Document Root
- Using Access Log Files and Server Log Files

HTTP Listeners

Connections between the server and clients happen on an HTTP listener, also called a listen socket. Each HTTP listener you create has an IP address, a port number, a return server name, and a default virtual server. If you want an HTTP listener to listen on all configured IP addresses on a given port for a machine, use 0.0.0.0, any, ANY, or INADDR_ANY for the IP address. For more information on creating and configuring HTTP Listeners, see "Creating and Configuring HTTP Listeners" on page 377.

When you install the unbundled version of Sun ONE Application Server, one HTTP listener, `http-listener-1`, is created automatically. This HTTP listener uses the IP address 0.0.0.0 and the port number you specified as your HTTP server port number during installation (the default is 80, or 1024 on UNIX if you are not installing as root). You cannot delete the default HTTP listener. If you are using multiple virtual servers, you can either use the default HTTP listener for all virtual servers, or create multiple HTTP listeners.

When you use the Solaris 9 bundled Sun ONE Application Server, your HTTP listener is created when you create the server instance. It has the IP address 0.0.0.0 and the port number you specified when you created your instance.

Since an HTTP listener is a combination of IP address and port number, you can have multiple HTTP listeners with the same IP address and different port numbers, or with different IP addresses and the same port number. For example, you could have 1.1.1.1:81 and 1.1.1.1:82. Additionally, you could have 1.1.1.1:81 and 1.2.3.4:81, as long as your machine is configured to respond to both these addresses. However, if you use the 0.0.0.0 IP address, which listens on all IP addresses on a port, you cannot set up HTTP listeners for additional IP addresses that listen on the same port for a specific IP address. For example, if you have an HTTP listener using 0.0.0.0:80 (all IP addresses on port 80) you cannot also create an HTTP listener which uses 1.2.3.4:80.

Each HTTP listener also has a default virtual server, which is the server to which it routes requests if it can't connect to the virtual server specified in the request.

In addition, you specify the number of acceptor threads (sometimes called accept threads) in the HTTP listener. Accept threads are threads that wait for connections. The threads accept connections and put them in a queue where they are then picked up by worker threads. Ideally, you want to have enough accept threads so that there is always one available when a new request comes in, but few enough so that they do not provide too much of a burden on the system. The default is 1. A good rule is to have one accept thread per CPU on your system. You can adjust this value if you find performance suffering.

You also specify whether security is enabled for an HTTP listener and what kind of security you are using (for example, what kind of SSL and which ciphers).

Virtual Servers

To create a virtual server you must first decide what kind of virtual server you want. You can have an IP-address-based virtual server, or a URL-host-based virtual server. To create a virtual server, all you need to specify is a virtual server ID, one or more HTTP listeners, and one or more URL hosts.

This section includes the following topics:

- Types of Virtual Servers
- IP-Address-Based Virtual Servers
- URL-Host-Based Virtual Servers
- Default Virtual Server

Types of Virtual Servers

All virtual servers have a URL host specified. However, the virtual server may also be associated with an IP address based on an HTTP listener. If the virtual server's HTTP listener listens on a specific IP address, the virtual server is called an IP-address-based virtual server.

If several virtual servers listen on the same IP address, they are distinguished by the URL host, and are URL-host-based virtual servers.

When a new request comes in, the server determines which virtual server to send it to based on the IP address or the value in the Host header. It evaluates the IP address first. For more information, see "Virtual Server Selection for Request Processing" on page 374.

IP-Address-Based Virtual Servers

In order to have multiple IP addresses on a single computer, you must either map them through the operating system or provide additional cards. To set up multiple IP addresses through the operating system, use the Network Control Panel (Windows) or the `ifconfig` utility (UNIX). Please note that directions for using `ifconfig` vary from platform to platform. Consult your operating system documentation for more information.

You create an IP-address-based virtual server by creating an HTTP listener that listens on a specific IP address. You then associate a virtual server as the default virtual server for the HTTP listener. For more information on ways to deploy virtual servers, see "Deploying Virtual Servers" on page 387.

URL-Host-Based Virtual Servers

You can set up URL-host-based virtual servers by giving them unique URL hosts. The contents of the Host request header directs the server to the correct virtual server.

For example, if you want to set up virtual servers for customers *aaa*, *bbb*, and *ccc*) so that each customer can have an individual domain name, you first configure DNS to recognize that each customer's URL (`www.aaa.com, www.bbb.com`, `www.ccc.com`) resolves to the IP address of the HTTP listener you are using. You then set the URL hosts for each virtual server to the correct setting (for example, `www.aaa.com`). Note that you map hosts to IP addresses in the `/etc/hosts` file.

You can have any number of these URL-host-based virtual servers associated with an HTTP listener.

Because URL-Host-based virtual servers use the Host request header to direct the user to the correct page, not all client software works with them. Older client software that does not support the HTTP Host header won't work. These clients will receive the default virtual server for the HTTP listener.

Default Virtual Server

URL-Host-based virtual servers are selected using the Host request header. If the end user's browser does not send the Host header, or if the server cannot find the specified Host header, the default virtual server for the HTTP listener services the request.

Also, for IP-address-based virtual servers, if Sun ONE Application Server cannot find the specified IP address, the default virtual server for the HTTP listener services the request. You can configure the default virtual server to send an error message, or server pages from a special document root.

NOTE Do not confuse the default virtual server for an HTTP listener with the default virtual server created when you install the server. The default virtual server is the virtual server for the default application server instance. The default virtual server for an HTTP listener is any virtual server you designate as the default.

You specify a default virtual server when you create an HTTP listener. You can always change the default virtual server.

The obj.conf File

By default, each virtual server has a separate `obj.conf` file where virtual server settings are stored. When you change settings through the Administration interface or command-line interface, those changes are made automatically in the configuration files, including the virtual server's `obj.conf` file. All `obj.conf` files are located in the *instance_dir*/config directory. Whenever this guide refers to "the obj.conf file," it refers to all `obj.conf` files or to the `obj.conf` file for the virtual server being described.

The file named `obj.conf` that lacks a prefix is a template that Sun ONE Application Server uses to create `obj.conf` files for each virtual server. Editing this file does not affect any existing virtual servers, but does affect any subsequently created virtual servers. For more information on editing the `obj.conf` file directly, see the *Sun ONE Application Server Administrator's Configuration File Reference.*

By default, each active obj.conf file is named *virtual_server_name*-obj.conf. Because the default virtual server for a server instance is named after the instance, when you first create a server instance, its obj.conf file is named *instance_name*-obj.conf. Editing one of these files directly or through the Administration interface changes the configuration of a virtual server.

Virtual Server Selection for Request Processing

Before the server can process a request, it must accept the request via an HTTP listener, then direct the request to the correct virtual server. This section discusses how the virtual server is determined.

- If the HTTP listener is configured to only a default virtual server, that virtual server is selected.
- If the HTTP listener has more than one virtual server configured to it, the request Host header is matched to the hosts attribute of a virtual server. If no Host header is present or no hosts attribute matches, the default virtual server for the HTTP listener is selected.

If a virtual server is configured to an SSL HTTP listener, its hosts attribute is checked against the subject pattern of the certificate at server startup, and a warning is generated and written to the server log if they don't match.

After the virtual server is determined, the Sun ONE Application Server executes the virtual server's obj.conf file. For details about how the server decides which directives to execute in obj.conf, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

Document Root

The document root (sometimes called the primary document directory) is the central directory that contains all the virtual server's files you want to make available to remote clients.

The document root directory provides an easy way to restrict access to the files on a virtual server. It also makes it easy to move documents to a new directory (perhaps on a different disk) without changing any of the URLs because the paths specified in the URLs are relative to the primary document directory.

For example, if your document directory is *install_dir*/`docs`, a request such as `http://www.sun.com/products/info.html` tells the server to look for the file in *install_dir*/`docs/info.html`. If you change the document root (that is, you move all the files and subdirectories), you only have to change the document root that the virtual server uses, instead of mapping all URLs to the new directory or somehow telling clients to look in the new directory.

The document root for your default Sun ONE Application Server instance (server1) becomes the document root for the virtual servers created within the server1 application server instance. You can override that directory for each virtual server you create.

Using Sun ONE Application Server Features with Virtual Servers

Sun ONE Application Server has many features, such as SSL and access control, that you can use with virtual servers. The following sections describe the features and provide information on where to look for more information.

This section includes the following topics:

- Using SSL with Virtual Servers
- Using Access Log Files and Server Log Files
- Using Access Control with Virtual Servers
- Using CGIs with Virtual Servers

Using SSL with Virtual Servers

If you want to use SSL on a virtual server, in most cases you use an IP-address-based virtual server. The customary port is 443. It is difficult to use SSL on a URL-host-based virtual server because Sun ONE Application Server must read the request before determining which URL host to send the request to. Once the server reads the request, the initial handshake, where security information is exchanged, has already happened.

The only exception is when URL-Host-based virtual servers all have the same SSL configuration, including the same server certificate, using “wildcard certificates.” For more information, see the *Sun ONE Application Server Administrator’s Guide to Security*.

One way to implement SSL with virtual servers is to have two HTTP listeners, one using SSL and listening to port 443, and one that is not using SSL. A user would typically access the virtual server through the non-SSL HTTP listener. When the need to have secure transactions arises, users could click a button on the web page to start initiating secure transactions. After that, the requests go through the secure HTTP listener.

Because SSL transactions are much slower than non-SSL transactions, this design limits the SSL transactions to only the ones that are necessary. Faster, non-SSL connections are used the rest of the time.

For more information on setting up and using security with you Sun ONE Application Server and virtual servers, see the *Sun ONE Application Server Administrator's Guide to Security*. For a diagram of a sample SSL configuration with virtual servers, see "Example 2: Secure Server" on page 388.

Using Access Log Files and Server Log Files

The access log file is the file where HTTP accesses to the virtual server are logged. When you create a new virtual server, by default the access log file is the same log file as the application server instance. In many cases you will want each individual virtual server to have its own log file. To set this up, you can change the log path for each virtual server. If you want to keep all virtual server accesses logged to the same access log file, you can change the logging settings for the server instance so that the virtual server ID is included in the log file. For more information on changing the application server instance's logging, see Chapter 5, "Using Logging."

The server log file is the file where informational messages and errors are logged. When you create a new virtual server, its log file by default is the same as the log file for the application server instance. You can change the log file for each virtual server.

Using Access Control with Virtual Servers

With virtual servers you have the ability to set up access control on a per virtual server basis. You can even configure it so that each virtual server can have user and group authentication using an LDAP database. For more information, see the *Sun ONE Application Server Administrator's Guide to Security*.

Using CGIs with Virtual Servers

You can use CGIs on virtual servers. You need to set up the directories where CGIs will be stored on each virtual server and set a file type for the CGI. For more information on CGIs, see the *Sun ONE Application Server Developer's Guide to Web Applications*.

Creating and Configuring HTTP Listeners

Before the server can process a request, it must accept the request via an HTTP listener, then direct the request to the correct virtual server. One HTTP listener, `http-listener-1`, is created automatically when a server instance is created (either during installation or later). This HTTP listener uses the IP address 0.0.0.0 and the port number you specified as your application server port number. You cannot delete the default HTTP listener.

This section covers the following topics:

- Creating an HTTP Listener
- Editing HTTP Listener Settings
- Deleting an HTTP Listener

Creating an HTTP Listener

To create an HTTP listener using the Administration interface:

1. In the left pane, for the application server instance, open HTTP Server.
2. Click HTTP Listeners.
3. Click New.

4. Fill in the fields.

   HTTP listeners must have a unique combination of port number and IP address. You can use either IPV4 or IPV6 addresses. If you want to create an HTTP listener for IP-address-based virtual servers, specify a particular IP address at for the HTTP listener.

   The Return Server Name field specifies the host name in the URLs the server sends to the client. This affects URLs the server automatically generates; it doesn't affect the URLs for directories and files stored in the server. This name should be the alias name if your server uses an alias.

   The default virtual server is the virtual server that will answer requests for the HTTP listener if no other virtual server is found first. For more information, see "Virtual Server Selection for Request Processing" on page 374.

   You must enable the HTTP listener before it can accept requests.

   You can also enable security and configure advanced properties for this HTTP listener. To specify IPV6, use the value inet6 in the Family field. If this value is inet6, IPv4 addresses are prefixed with ::ffff: in the server log.

5. Click OK.

Please note that you must enter an existing virtual server in the default virtual server field when you create an HTTP listener. You can use the virtual server created with the server instance, and then go back and change it after you've created additional virtual servers, if you like.

To create an HTTP listener using the command-line interface, use the `asadmin` utility's `create-http-listener` command. To get a list of all created HTTP listeners, use the command `list-http-listeners`.

To create an HTTP Listener, use the following syntax:

```
asadmin create-http-listener --user username [--password password]
[--host hostname] [--port adminport] [--secure | -s] [--passwordfile
file_name] --address address [--instance instancename] --listenerport
listener_port --defaultvs virtual_server --servername server_name [--family
family] [--acceptorthreads acceptor_threads] [--blockingenabled
blocking_enabled] [--securityenabled security_enabled] [--enabled enabled]
listener_id
```

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Editing HTTP Listener Settings

To edit HTTP listener settings using the Administration interface:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open HTTP Listeners.
3. Click the HTTP listener you want to edit.
4. Make the desired changes and click Save.

For more information, see the online help.

You can also edit an HTTP listener using the `asadmin` utility in the command-line interface. Use the `get` command to get the current settings, and the `set` command to set them to new values.

To get the values of all the attributes of an HTTP listener:

```
asadmin> get server_instance.http-listener.http_listener_name.*
```

For example, to get the values for the default HTTP listener:

```
asadmin> get server1.http-listener.http-listener-1.*
```

To set the value of any attribute:

```
asadmin> set server_instance.http-listener.http_listener_name.attribute_name=value
```

For example, to set the attribute `defaultVirtualServer` to `server2` for `http-listener-1`:

```
asadmin> set
server1.http-listener.http-listener-1.defaultVirtualServer=server2
```

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, “Using the Command Line Interface.”

Deleting an HTTP Listener

To delete an HTTP listener using the Administration interface:

1. In in the left pane, for the application server instance, open HTTP Server.
2. Click HTTP Listeners.
3. Click the checkbox next to the HTTP listener you want to delete.
4. Click Delete.

To delete an HTTP listener using the command-line interface, use the asadmin utility's delete-http-listener command, using the following syntax:

```
asadmin delete-http-listener ---user username [--password password]
[--host hostname] [--port adminport] [--secure | -s] [--passwordfile
file_name] --instance instance httplistener_id
```

For more information on command syntax, see the command-line interface help. For more information on using asadmin, see Appendix A, "Using the Command Line Interface."

Creating and Configuring Virtual Servers

Once you have set up an HTTP Listener, you can create and use virtual servers.

This section covers the following topics:

- Creating a Virtual Server
- Editing Virtual Server Settings
- Deleting a Virtual Server

Creating a Virtual Server

To create a virtual server using the Administration interface:

1. In the left pane, for the application server instance, open HTTP Server.
2. Click Virtual Servers.
3. Click New.
4. Fill in the required fields and any optional fields.
5. Click Save.

To create a virtual server using the command-line interface, use the asadmin utility's create-virtual-server command, using the following syntax:

```
asadmin create-virtual-server --user username ---user username
[--password password] [--host hostname] [--port adminport] [--secure |
-s] [--passwordfile file_name] [--instance instancename] --hosts hosts
--mime mime_types_file [--httplisteners http-listeners] [--defaultwebmodule
default_web_module] [--configfile config_file] [--defaultobj default_object]
[--state state] [--acls acls] [--acceptlang accept_language] [--logfile
logfile] [--property (name=value)[:name=value]*] virtual_server_id
```

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, “Using the Command Line Interface.”

When you create a virtual server, you can enter settings of the following types:

- Required Settings
- Optional General Settings
- Web Application Settings
- CGI Settings
- HTTP Quality of Service Settings

Required Settings

The required setting for a virtual server include the name (ID), and the URL host or hosts.

You also must specify a MIME types file. The MIME types file contains the mappings of file extensions to types of files. For example, the MIME types file is where you can specify that all files ending .cgi be treated as CGI files.

You don’t need to create a separate MIME types file for each virtual server. Instead, you create as many MIME types files as you need and associate them with a virtual server.The default MIME types file is called `mime1` and the file name is `mime.types`.

For more information on MIME types files, see “Configuring MIME Types” on page 367.

Optional General Settings

In addition to the required fields, you can also set optional fields.

HTTP listener

The HTTP listener handles the connection to the virtual server. You must specify one in order to have remote clients access the virtual server.

ACLs

The access control list (ACL) applied to the virtual server. For more information, see the *Sun ONE Application Server Administrator’s Guide to Security*.

Accept Language Header

When clients contact a server using HTTP 1.1, they can send header information describing the languages they accept. You can configure your server to parse this language information.

For example, if you store documents in Japanese and English, you could choose to parse the accept language header. When clients that have Japanese as the accept language header contact the server, they receive the Japanese version of the page. When clients that have English as the accept language header contact the server, they receive the English version.

If you do not support multiple languages, you should not parse the accept language header.

State

This state is the virtual server's state, which is independent of whether the application server instance is On or Off. If a virtual server's state displayed on this page is On, the virtual server can only accept requests if the application server instance is On as well.

This is true of the default virtual server for the default application server instance as well. If you turn off your application server instance, your default virtual server is still set to On, but will not accept connections.

Valid states are On, Off, or Disabled. A virtual server is able to accept connections if it is set to On

You cannot turn off or disable the default virtual server for the application server instance.

Log Files

The log file, (also known as the server log file) is the file where informational messages and errors are logged. The access log file is the file where HTTP accesses to the virtual server are logged.

Document Root

The document root (sometimes called the primary document directory) is the central directory that contains all the virtual server's files you want to make available to remote clients. For more information, see "Document Root" on page 374.

Web Application Settings

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.

Sun ONE Application Server 7 supports the Servlet 2.3 API specification, which allows servlets and JSPs to be included in web applications. In addition, Sun ONE Application Server 7 supports SHTML and CGI, which are non-J2EE application components.

When you create a virtual server, you specify a default web module for the virtual server. The default web module responds to all requests that cannot be resolved to other web modules deployed to the virtual server. If you don't specify a default web module, the web module that has an empty context root is used. If there's no web module with an empty context root, a system default web module is created and used.

When you deploy a web application, you specify a virtual server. Once you have deployed a web application, it appears in the list of available web modules to choose as the default web module for a virtual server. When you specify a web module as the default web module for a virtual server, the virtual server is automatically added to the web application's list of virtual servers.

CGI Settings

The CGI settings you set when you create a virtual server govern the user and group CGI programs run as, the directory to change to (chroot) before CGI execution begins, and the directory to change to after the chroot.

On UNIX you can also set nice, an increment that determines a 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).

HTTP Quality of Service Settings

Quality of service refers to the performance limits you set for a virtual server. For example, an ISP might want to charge different amounts of money for virtual servers depending on how much bandwidth allowed them. These settings can either be enforced (that is, only the specified bandwidth and maximum number of connections will be allowed) or not enforced. If the settings are not enforced, a message is logged to the log file when the limits are exceeded. For more information, see "Administering the Transaction Service Using the CLI" on page 153.

In addition to changing these settings through the Administration interface, you can use the command-line interface's `asadmin` utility. To configure quality of service using the command-line interface's `asadmin` utility, use the following commands:

- `create-http-qos`
- `delete-http-qos`

These commands use the following syntax:

```
asadmin create-http-qos --user username [--password password] [--host
hostname] [--port adminport] [--secure | -s] [--passwordfile file_name]
[--virtualserver virtual_server_id] [--bwlimit bandwidth_limit]
[--enforcebwlimit enforce_bandwidth_limit] [--connlimit connection_limit]
[--enforceconnlimit enforce_connection_limit] instance_name
```

```
asadmin delete-http-qos --user username [--password password] [--host
hostname] [--port adminport] [--secure | -s] [--passwordfile file_name]
[--virtualserver virtual_server_id] instance_name
```

If you specify a virtual server, these commands create or delete quality of service information for that virtual server. If you do not specify a virtual server, the command affects the server instance.

For more information on command syntax, see the command-line interface help. For more information on using `asadmin`, see Appendix A, "Using the Command Line Interface."

Editing Virtual Server Settings

Once you have set up your virtual servers, you can edit them. For information on editing virtual server settings, see the following topics:

- Editing General Settings Using the Administration Interface
- Editing General Settings Using the Command-Line Interface
- Editing CGI Settings
- Editing Document Handling Settings, Document Directories Settings, and HTTP/HTML Settings

Editing General Settings Using the Administration Interface

The virtual server's general settings are the ones you could set when you created the virtual server. To change them, follow these steps:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the virtual server you want to edit.
4. Make your desired changes.

   The areas you can change include quality of service settings, adding ACLs, content-related settings such as the document root and the accept language header, CGI-related settings such as the user, group, nice, and chroot settings, and the default web module.
5. Click Save.

For more information on some of these settings, see "Creating and Configuring Virtual Servers" on page 380. Also see the online help.

Editing General Settings Using the Command-Line Interface

You can also edit these setting using the `asadmin` utility in the command-line interface. Use the `get` command to get the current settings, and the `set` command to set them to new values.

To get all the attributes from a virtual server, use the following syntax:

```
asadmin> get instance_name.virtual-server.vserver_id.*
```

For example:

```
asadmin> get server1.virtual-server.vs1.*
```

If you want to get all the attributes for the application server instance `server1`, use the following syntax:

```
asadmin> get server1.virtual-server.server1.*
```

To set an attribute, for example, the accept language header, use the following syntax:

```
asadmin> set
server1.virtual-server.server1.virtualserver.acceptLanguage=false
```

NOTE You can user the command-line interface to set values for all the fields on the General page. However, you cannot use the command-line interface to set values for the fields on the pages on other tabs, for example, the pages on the CGI tab.

For more information on command syntax, see the command-line interface help. For more information on using asadmin, see Appendix A, "Using the Command Line Interface."

Editing CGI Settings

For information on editing CGIs, see the *Sun ONE Application Server Developers Guide to Web Applications*.

Editing Document Handling Settings, Document Directories Settings, and HTTP/HTML Settings

For information on changing these settings, see Chapter 16, "Managing Virtual Server Content."

Deleting a Virtual Server

To delete a virtual server:

1. In the Administration interface, in the left pane, for the application server instance, open HTTP Server.
2. Click Virtual Servers.
3. Click the checkbox next to the virtual server you want to delete.
4. Click Delete.

   You cannot delete all virtual servers using the Administration interface.

To delete a virtual server using the command-line interface, use the asadmin utility's delete-virtual-server command.

Usage is as follows:

```
asadmin delete-virtual-server --user username [--password password]
[--host hostname] [--port adminport] [--secure | -s] [--passwordfile
file_name] --instance instance virtualserver_id
```

For more information on command syntax, see the command-line interface help. For more information on using asadmin, see Appendix A, "Using the Command Line Interface."

Deploying Virtual Servers

Sun ONE Application Server's virtual server architecture is very flexible. An application server instance can have any number of HTTP listeners, both secure and non-secure. You can associate any number of virtual servers with these HTTP listeners. You can have both IP-address-based and URL-host-based virtual servers.

Every virtual server can (but does not have to) have its own list of ACLs, its own `mime.types` file, and its own set of Java Web Applications.

This design gives you maximum flexibility to configure the server for a variety of applications. The following examples discuss some of the possible configurations available for Sun ONE Application Server.

- Example 1: Default Configuration
- Example 2: Secure Server
- Example 3: Intranet Hosting
- Example 4: Mass Hosting

Example 1: Default Configuration

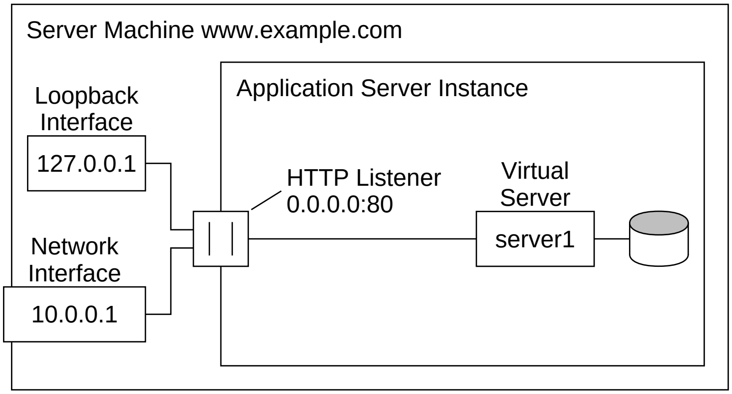
The default configuration is one application server instance. This application server instance has just one HTTP listener listening on port 80, 1024, or whatever you selected, of any IP address to which your computer is configured.

Some mechanism in your local network establishes a name-to-address mapping for each of the addresses to which your computer is configured. In the following example, the computer has two network interfaces: the loopback interface (the interface that exists even without a network card) on address 127.0.0.1, and an ethernet interface on address 10.0.0.1.

The name example.com is mapped to 10.0.0.1 via DNS. The HTTP listener is configured to listen on port 80 on any address to which that machine is configured ("0.0.0.0:80").

As there are no IP-address-based virtual servers in the default configuration, the only HTTP listener is the default one. All connections pass through to virtual server server1.

Figure 15-1 Default Configuration



DNS

www.example.com	10.0.0.1

In this configuration, connections to the following reach the server and are served by virtual server VS1

- http://127.0.0.1/ (initiated on example.com)
- http://localhost/ (initiated on example.com)
- http://example.com/
- http://10.0.0.1/

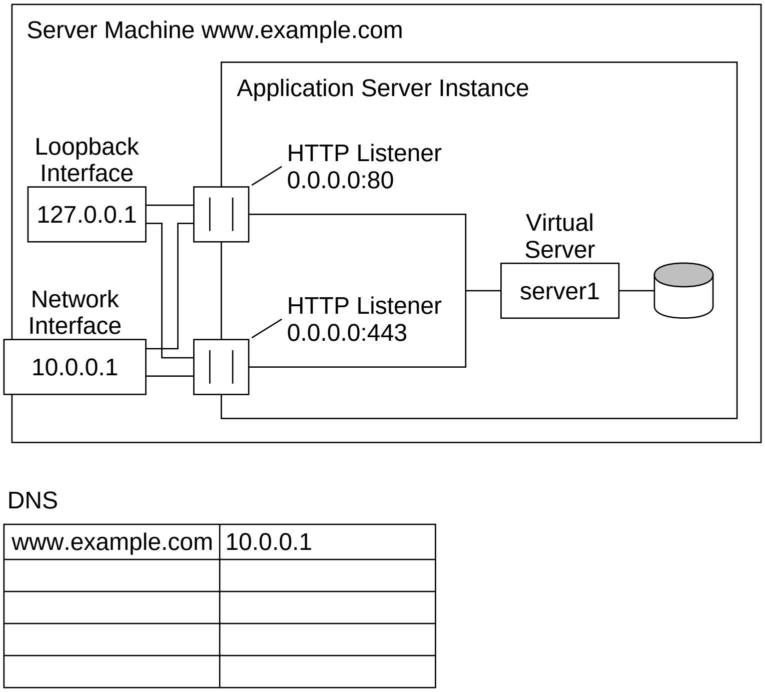
Use this configuration for traditional HTTP server use. You do not need to add additional virtual servers or HTTP listeners. You configure the settings of the server by changing the settings for server1.

Example 2: Secure Server

If you want to use SSL in the default configuration, you can simply change the HTTP listener to secure mode.

You can also add a new secure HTTP listener configured to 0.0.0.0:443 and associate server1 to the new HTTP listener. The virtual server now has HTTP listeners, one that uses the secure HTTP listener, and one that doesn't. Now your server will serve the same content both with and without SSL, i.e. http://example.com/ and https://example.com/ deliver the same content.

Figure 15-2 Secure Server



www.example.com	10.0.0.1

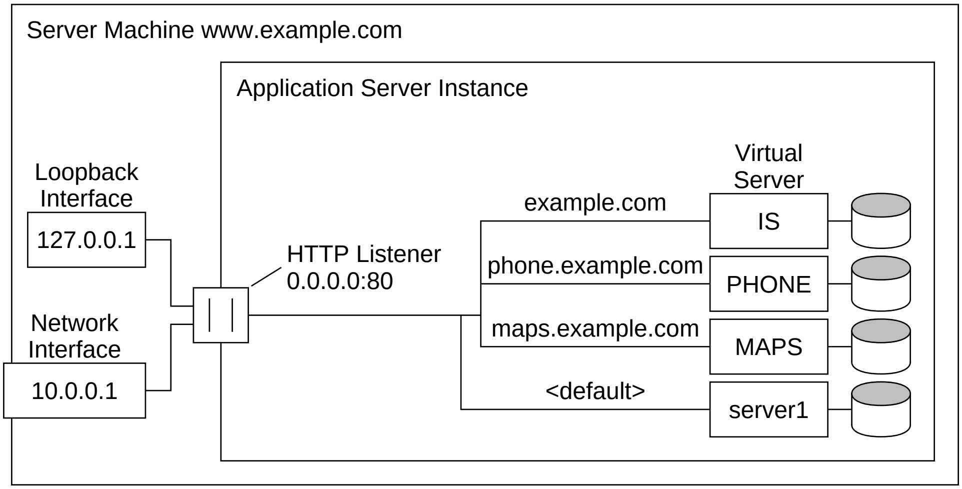
Please note that the SSL parameters are attached to the HTTP listener.

Example 3: Intranet Hosting

A more complex configuration of the Sun ONE Application Server is one in which the server hosts a few virtual servers for an intranet deployment. For example, you have three internal sites where employees can look up other users' phone numbers, look at maps of the campus, and track the status of their requests to the Information Services department. Previously (in this example), these sites were hosted on three different computers that had the names phone.example.com, maps.example.com and is.example.com mapped to them.

To minimize hardware and administrative overhead, you want to consolidate all three sites into one application server living on the machine example.com. You could set this up in two ways: using URL-host-based or IP-address-based virtual servers. Both have distinct advantages and disadvantages.

Figure 15-3 Intranet Hosting Using URL-Host-Based Virtual Servers



DNS

www.example.com	10.0.0.1
is.example.com	10.0.0.1
phone.example.com	10.0.0.1
maps.example.com	10.0.0.1

While URL-host-based virtual servers are easy to set up, they have the following disadvantages:

- Supporting SSL in this configuration requires non-standard setup using wildcard certificates. For more information see the *Sun ONE Application Server Administrator's Guide to Security*.
- URL-host-based virtual servers don't work with legacy HTTP clients

The advantages to IP-address-based virtual servers are:

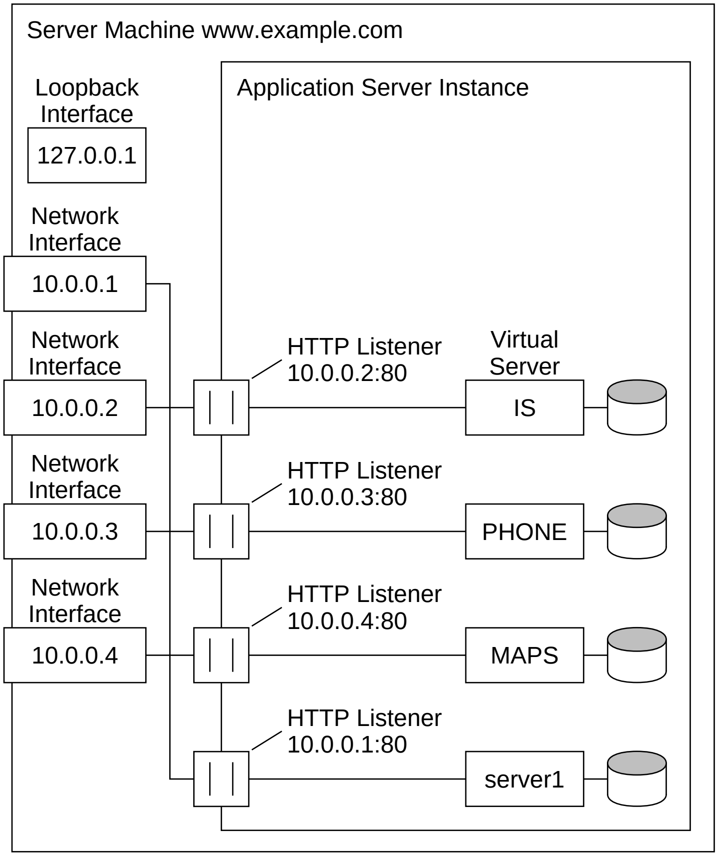
- They work with older clients that do not support the HTTP/1.1 Host header.
- Providing SSL support is straightforward.

The disadvantages are:

- They require configuration changes on the host computer (configuration of real or virtual network interfaces)
- They don't scale to configurations with thousands of virtual servers

Both configurations require setting up name-to-address mappings for the three names. In the IP-address-based configuration, each name maps to a different address. The host machine must be set up to receive connections on all these addresses. In the URL-host-based configuration, all names can map to the same address, the one the machine had originally.

Figure 15-4 Intranet Hosting Using IP-Address-Based Virtual Servers



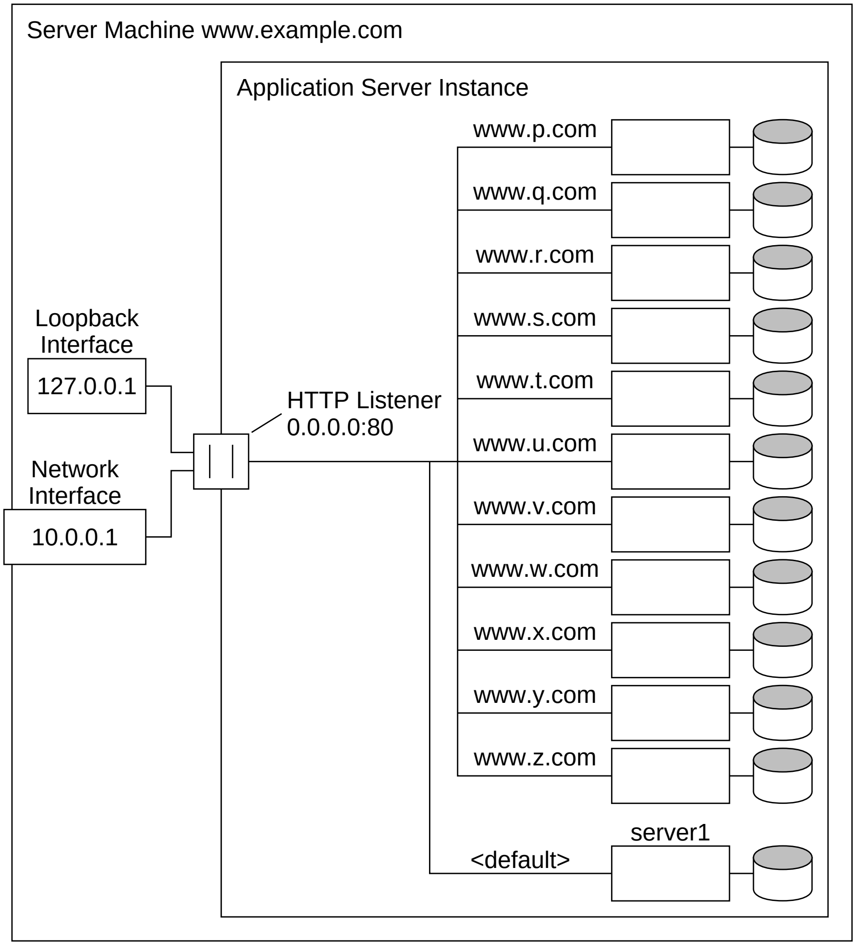
DNS

www.example.com	10.0.0.1
is.example.com	10.0.0.2
phone.example.com	10.0.0.3
maps.example.com	10.0.0.4

Example 4: Mass Hosting

Mass hosting is a configuration in which you enable many low-traffic virtual servers. For example, an ISP that hosts many low-traffic personal home pages would fall into this category. The virtual servers are usually URL-host-based.

Figure 15-5 Mass Hosting



DNS

www.example.com	10.0.0.1
www.p.com	10.0.0.1
www.q.com	10.0.0.1
www.r.com	10.0.0.1
. . .	
www.z.com	10.0.0.1

Notice that the default virtual server, `server1`, still exists.

Chapter 16

Managing Virtual Server Content

This chapter describes how you can configure and manage the files served by virtual servers.

This chapter includes the following topics:

Changing the Document Root

The document root is the central directory where you store all the files you want to make available to remote clients.

When you add a virtual server, you specify a document root with an absolute path. For more information about the document root and how it is used, see "Document Root" on page 374.

To use the Administration interface to change the document root to use a different path:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the General Tab.
5. Enter an absolute directory path in the Document Root field.

   You need to create this directory manually.
6. Click OK.

For more information, see the online help.

NOTE Typically, each virtual server has its own document root.

Setting Additional Document Directories

Most of the time, the documents for a virtual or server instance are in the document root. Sometimes, though, you may want to serve documents from a directory outside of the document root. You can do this by setting additional document directories. By serving from a document directory outside of the document root, you can let someone manage a group of documents without giving them access to your primary document root.

To use the Administration interface to add an additional document directory:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.

4. Click the Doc Directories tab.
5. Click Additional Doc Directories.
6. Choose the URL prefix to map.

   Clients send this URL to the server when they want documents.
7. Specify the directory to map those URLs to.
8. Click OK.

To for more information, see the online help.

You should restrict access to additional document directories so that users cannot write to them.

Enabling Remote File Manipulation

When you enable remote file manipulation, clients are able to upload files, delete files, create directories, remove directories, list the contents of a directory, and rename files on your server. The virtual servers' configuration file `obj.conf` contains the commands that are activated when you enable remote file manipulation. By activating these commands, you allow remote browsers to change a server's documents. You should use access control to restrict write access to these resources to prevent unauthorized tampering.

Note that enabling remote file manipulations should have no effect on using content management systems such as Microsoft Frontpage.

UNIX: You must have the correct permissions for your files or this function will not work; that is, the document root user must be the same as the server user.

To use the Administration interface to enable remote file manipulation:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Directories tab.
5. Click Remote File Manipulation.
6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a virtual server.
7. Choose to activate remote file manipulation.

8. Click OK.

For more information, see the online help.

Using htaccess

The `htaccess` files are dynamic configuration files that store a subset of configuration options. You can use `htaccess` files in combination with the Sun ONE Application Server standard access controls (standard access controls are always applied before any `htaccess` access controls).

For information on using htaccess, see the *Sun ONE Application Server Administrator's Guide to Security*.

Restricting Symbolic Links (UNIX)

You can limit the use of the file system links in your server. File system links are references to files stored in other directories or file systems. The reference makes the remote file as accessible as if it were in the current directory. There are two types of file system links:

- Hard links—A hard link is really two filenames that point to the same set of data blocks; the original file and the link are identical. For this reason, hard links cannot be on different file systems.
- Symbolic (soft) links—A symbolic link consists of two files, an original file that contains the data, and another that points to the original file. Symbolic links are more flexible than hard links. Symbolic links can be used across different file systems and can be linked to directories.

For more information about hard and symbolic links, see your UNIX system documentation.

File system links are an easy way to create pointers to documents outside of the primary document directory and anyone can create these links. For this reason you might be concerned that people might create pointers to sensitive files (for example, confidential documents or system password files).

To use the Administration interface to restrict symbolic links:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.

3. Click the name of the virtual server you want to edit.
4. Click the Doc Directories tab.
5. Click Symbolic Links.
6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a virtual server.
7. Choose whether to enable soft and/or hard links and the directory to start from.
8. Click OK.

For more information, see the online help.

Customizing User Public Information Directories (UNIX)

Sometimes users want to maintain their own web pages. You can configure public information directories that let all the users on a server create home pages and other documents without your intervention.

NOTE Though the User Document Directories page appears in the Administration interface for Windows systems, the feature is not available.

With this system, clients can access your server with a certain URL that the server recognizes as a public information directory. For example, suppose you choose the prefix ~ and the directory `public_html`. If a request comes in for `http://www.sun.com/~jdoe/aboutjane.html`, the server recognizes that `~jdoe` refers to a users' public information directory. It looks up `jdoe` in the system's user database and finds Jane's home directory. The server then looks at `~/jdoe/public_html/aboutjane.html`.

This section contains the following topics:

- Configuring Public Information Directories
- Restricting Content Publication
- Loading the Entire Password File on Startup

Configuring Public Information Directories

To use the Administration interface to configure your virtual server to use public directories:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Handling tab.
5. Click User Doc Directories.
6. Choose a user URL prefix.

   The usual prefix is ~ because the tilde character is the standard UNIX prefix for accessing a user's home directory.

7. Choose the subdirectory in the user's home directory where the server looks for HTML files.

   A typical directory is `public_html`.

8. Designate the password file.

   The server needs to know where to look for a file that lists users on your system. The server uses this file to determine valid user names and to find their home directories. If you use the system password file for this purpose, the server uses standard library calls to look up users. Alternatively, you can create another user file to look up users. You can specify that user file with an absolute path.

   Each line in the file should have this structure (the elements in the `/etc/passwd` file that aren't needed are indicated with `*`):

   ```
   username:*:*:groupid:*:homedir:*
   ```

9. Choose whether to load the password database at startup.

   For more information, see "Loading the Entire Password File on Startup" on page 401.

10. Click OK.

For more information, see the online help.

Another way to give users separate directories is to create a URL mapping to a central directory that all of your users can modify.

Restricting Content Publication

In some situations a system administrator may want to restrict what user accounts are able to publish content via user document directories. To restrict a user's publishing, add a trailing slash to the user's home directory path in the `/etc/passwd file`:

```
jdoe::1234:1234:John Doe:/home/jdoe:/bin/sh
```

becomes:

```
jdoe::1234:1234:John Doe:/home/jdoe/:/bin/sh
```

After you make this modification, Sun ONE Application Server will not serve pages from this user's directory. The browser requesting the URI receives a "404 File Not Found" error and a 404 error will be logged to the access log.

If, at a later time, you decide to allow this user to publish content, remove the trailing slash from the `/etc/passwd` entry, then restart the Application Server Instance.

Loading the Entire Password File on Startup

You also have the option of loading the entire password file on startup. If you choose this option, the server loads the password file into memory when it starts, making user lookups much faster. If you have a very large password file, however, this option can use too much memory.

Setting the Document Preferences

This section contains the following topics:

- Entering an Index Filename
- Selecting Directory Indexing
- Specifying a Server Home Page
- Specifying a Default MIME Type

To use the Administration interface to set the document preferences, follow these steps:

1. In the left pane, for the application server instance, open HTTP Server.

2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Handling tab.
5. Click Doc Preferences.
6. Choose the appropriate field values, as discussed in the following sections.
7. Click OK.

The preferences you can set are discussed more fully in the sections that follow. For additional information, see the online help.

Entering an Index Filename

If a document name is not specified in the URL the server automatically displays the index file. The default index files are `index.html` and `home.html`. If more than one index file is specified, the server looks in the order in which the names appear in this field until one is found. For example, if your index filenames are `index.html` and `home.html`, the server looks for `index.html` and if it doesn't find it looks for `home.html`.

Selecting Directory Indexing

A document directory will probably have several subdirectories. For example, there might be a directory called `products`, another called `people`, and so on. It's often helpful to let clients access an overview (or index) of these directories.

The server indexes directories by searching the directory for an index file called `index.html` or `home.html`, which is a file you create and maintain as an overview of the directory's contents. For more information, see "Entering an Index Filename" on page 402. You can specify any file as an index file for a directory by naming it one of these default names, which means you can also use a CGI program as an index.

If an index file isn't found, the server generates an index file that lists all the files in the document root.

CAUTION If your server is outside the firewall, turn off directory indexing to ensure that your directory structure and filenames are not accessible.

Specifying a Server Home Page

When end users first access the server, the first file they see is usually called a home page. Usually, this file has general information about your server and links to other documents.

By default, the server finds the index file specified in the Index Filename field in the Document Preferences page and uses that for the home page. However, you can also specify a file to use as the home page.

Specifying a Default MIME Type

When a document is sent to a client, the server includes a section that identifies the document's type, so the client can present the document in the right way. However, sometimes the server can't determine the proper type for the document because the document's extension is not defined for the server. In those cases, a default value is sent.

The default is usually text/plain, but you should set it to the type of file most commonly stored on your server. Some common MIME types include the following:

- text/plain
- text/richtext
- image/jpeg
- application/x-tar
- application/x-gzip
- text/html
- image/tiff
- image/gif
- application/postscript
- audio/basic

Customizing Error Responses

You can specify a custom error response that sends a detailed message to clients when they encounter errors from your virtual server. You can specify a file to send or a CGI program to run.

For example, you can change the way the server behaves when it gets an error for a specific directory. If a client tries to connect to a part of your server protected by access control, you might return an error file with information on how to get an account.

Before you can enable a custom error response, you must create the HTML file to send or the CGI program to run in response to an error. After you do this, enable the response in the Administration interface.

To use the Administration interface to enable a customized error response:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Handling tab.
5. Click Error Responses.
6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a virtual server.
7. For each error code you want to change, specify the absolute path to the file or CGI that contains the error response.
8. Click OK.

For more information see the online help.

Changing the International Character Set

The character set of a document is determined in part by the language it is written in. You can override a client's default character set setting for a document, a set of documents, or a directory by selecting a resource and entering a character set for that resource.

Browsers can use the MIME type `charset` parameter in HTTP to change its character set. If the server includes this parameter in its response, the browser changes its character set accordingly. Examples are:

- `Content-Type: text/html;charset=iso-8859-1`
- `Content-Type: text/html;charset=iso-2022-jp`

The following `charset` names are specified in RFC 1700 (except for the names that begin with `x-`):

- `us-ascii`
- `iso-2022-jp`
- `iso-8859-1`
- `x-sjis`

- `x-euc-jp`
- `x-mac-roman`

Additionally, the following aliases are recognized for `us-ascii`:

- `ansi_x3.4-1968`
- `ansi_x3.4-1986`
- `ascii`
- `us`
- `cp367`
- `iso-ir-6`
- `iso_646.irv:1991`
- `iso646-us`
- `ibm367`

The following aliases are recognized for `iso_8859-1`:

- `latin1`
- `iso_8859-1:1987`
- `ibm819`
- `iso_8859-1`
- iso-ir-100
- `cp819`

To use the Administration interface to change the character set:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Handling tab.
5. Click International Characters.
6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a virtual server.
7. Set the character set for all or part of the server.

   If you leave this field blank, the character set is set to NONE.
8. Click OK.

For more information, see the online help.

Setting the Document Footer

You can specify a document footer, which can include the last-modified time, for all the documents in a certain section of the server. This footer works for all files except output of CGI scripts or parsed HTML (.shtml) files. If you need your document footer to appear on CGI-script output or parsed HTML files, enter your footer text into a separate file and add a line of code or another server-side include to append that file to the page's output.

To use the Administration interface to set the document footer, follow these steps:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the Doc Handling tab.
5. Click Doc Footer.
6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a virtual server.

   If you choose a directory, the document footer applies only when the server receives a URL for that directory or any file in that directory.
7. Specify the type of files that you want to have include the footer.
8. Specify the date format.
9. Type any text you want to have appear in the footer.

   The maximum number of characters for a document footer is 765. If you want to include the date the document was last modified, type the string :LASTMOD:.

For more information see the online help.

Configuring URL Forwarding

URL forwarding allows you to redirect document requests to another server. Forwarding URLs or redirection is a method for the server to tell a user that a URL has changed (for example, because you have moved files to another directory or server). You can also use redirection to seamlessly send a person who requests a document on one server to a document on another server.

For example, if you forward http://www.sun.com/info/movies to a prefix film.sun.com, the URL http://www.sun.com/info/movies redirects to http://film.sun.com/info/movies.

Sometimes you may want to redirect requests for all the documents in one sub-directory to a specific URL. For example, if you had to remove a directory because it was causing too much traffic, or because the documents were no longer to be served for any reason, you could direct a request for any one the documents to a page explaining why the documents were no longer available. For example, a prefix on /info/movies could be redirected to http://www.sun.com/explain.html.

To use the Administration interface to configure URL forwarding:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the HTTP/HTML tab.
5. Click URL Forwarding.
6. Type the URL prefix you want to redirect, and whether you want to redirect it to another prefix or to a static URL.
7. Click OK.

For more information see the online help.

Setting up Server-Parsed HTML

HTML is normally sent to the client exactly as it exists on disk without any server intervention. However, the server can search HTML files for special commands (that is, it can parse the HTML) before sending documents. If you want the server to parse these files and insert request-specific information or files into documents, you must first enable HTML parsing.

To use the Administration interface to set up HTML parsing:

1. In the left pane, for the application server instance, open HTTP Server.
2. Open Virtual Servers.
3. Click the name of the virtual server you want to edit.
4. Click the HTTP/HTML tab.

5. Click Parse HTML.

6. Choose Entire Server from the resource picker to apply your change to the whole virtual server, or navigate to a specific directory within a 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 file s 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 OK.

For more information on setting your server to accept parsed HTML, see the online help.

For more information on using server-parsed HTML, see the *Sun ONE Application Server Developer's Guide to Web Applications.*

Setting Cache Control Directives

Cache-control directives are a way for Sun ONE Application Server to control what information is cached by a proxy server. Using cache-control directives, you override the default caching of the proxy to protect sensitive information from being cached, and perhaps retrieved later. For these directives to work, the proxy server must comply with HTTP 1.1.

For more information HTTP 1.1, see the Hypertext Transfer Protocol--HTTP/1.1 specification (RFC 2068) at:

http://www.ietf.org/

To use the Administration interface to set cache control directives:

1. In the left pane, for the application server instance, open HTTP Server.

2. Open Virtual Servers.

3. Click the name of the virtual server you want to edit.

4. Click the HTTP / HTML tab.

5. Click Cache Control Directives

6. Fill in the fields. Valid values for the response directives are as follows:

   - **Public.** The response is cachable by any cache. This is the default.
   - **Private.** The response is only cachable by a private (non-shared) cache.
   - **No Cache.** The response must not be cached anywhere.
   - **No Store.** The cache must not store the request or response anywhere in nonvolatile storage.
   - **Must Revalidate.** The cache entry must be revalidated from the originating server.
   - **Maximum Age (sec).** The client does not accept a response that has an age greater than this age.

7. Click OK.

For more information see the online help.

Using Stronger Ciphers

For information on setting stronger ciphers, see the *Sun ONE Application Server Administrator's Guide to Security*.

Part 4

Appendixes

Appendix A

Using the Command Line Interface

This appendix provides instructions for using the command line interface (the `asadmin` utility), in singlemode (that is, you run one command at a time from the command prompt) at the system prompt, in multimode (that is, you can run multiple commands without needing to reenter environment-level information), and in scripts and programs. You can use the command line interface in place of the Administration interface screens.

This appendix contains the following sections:

- About the Command Line Interface
- Using asadmin
- Security Considerations
- Concurrent Access Considerations
- Command Reference

About the Command Line Interface

This section contains the following topics:

- About the asadmin Utility
- About Ant Tasks
- About Other Command Line Utilities

About the asadmin Utility

The `asadmin` utility performs all configuration and administration tasks. You can use this utility in place of using the Administration interface.

About Ant Tasks

Many developers use Ant to accelerate the development process for J2EE applications. Ant scripts leverage the `asadmin` utility for some tasks. Developers use the Ant tasks for building applications, deploying and undeploying modules and applications and for controlling the Sun ONE Application Server.

For more information on the Ant tasks, see the *Sun ONE Application Server Developer's Guid*e.

For more information on Ant, see the Jakarta Project site at http://jakarta.apache.org/ant/.

About Other Command Line Utilities

Sun ONE Application Server contains additional command line utilities. The following table lists the utilities and gives a brief description of each one.

Table A-1 Other Command-Line Utilities

Utility	Definition
appclient	Launches the Application Client Container and invokes the client application packaged in the application JAR file.
capture-schema	Gets the database schema and mapping information.
flexanlg	Generates statistics about your server.
htpasswd	Creates the user authentication files.
package-appclient	Packs the application client container libraries and jar files. For more information, see the *Sun ONE Application Server Developer's Guide to Clients*.
verifier	Validates the J2EE deployment descriptors with the DTDs. For more information, see the *Sun ONE Application Server Developer's Guide*.

Table A-1 Other Command-Line Utilities

Utility	Definition
wscompile	Takes the service definition interface and generates the client stubs or server-side skeletons; or generates a set of WSDL for the provided interface.
wsdeploy	Generates a deployable WAR file.

For more information on these utilities, see their online help.

Using asadmin

The `asadmin` utility has a set of commands for performing administrative tasks. You can use these commands to most of the tasks that you can perform using the Administration interface. You can find the `asadmin` utility at *install_dir*/bin and run it from there. On Windows, double-click the `asadmin.bat` file and the `asadmin` utility launches in a command window, running in multimode.

Please note that some HTTP-server-related properties and the Administration Server properties cannot be set using the command line; they must be set using the Administration interface. You can set all properties stored in the `server.xml` configuration file, but not the ones stored in `init.conf` or `obj.conf`. For more information on the configuration files, see the *Sun ONE Application Server Administrator's Configuration File Reference*.

For more information on individual commands, see "Command Reference" on page 433 and the help for the commands.

This section contains the following topics:

- Understanding the Command Syntax
- Using Singlemode and Multimode
- Using Interactive and Non-Interactive Options
- Using the Environment Commands
- Using the Password File Option
- Running asadmin Locally or Remotely
- Using Command Line Invocations
- Using Escape Characters

- Using get and set Commands
- Using Help
- Viewing Output and Errors

Understanding the Command Syntax

The `asadmin` utility has the following syntax:

asadmin command -*short-option argument* --*long-option argument operand*

Command

The command is the operation or task performed. The command is case-sensitive.

Options

Options modify how the utility performs a command. Options are case-sensitive. Notice that the short options have a single dash (-) in front of them, and the long options have two dashes (--) in front of them. For many options, you can use either a long or short form; for example, user can be either `--user` or `-u`. Some options are required and some are optional. Optional options are shown in brackets in a command's syntax. You must include all required options when you run a command or you receive an error message and the command does not execute.

For a list of available long and short option names available see the "Short and Long Options, Default Values, and Environment Variables" table in "Command Reference" on page 433.

Most options require argument values, for example --port *port_number*. The exceptions are boolean options, which toggle to switch a feature on or off and don't require argument values.

You can also save options in the environment variables. For more information, see "Using the Environment Commands" on page 419. For a complete list of the environment variable equivalent of options, see "Long and Short Option Formats, Default Values, and Environment Variable Equivalents" on page 464.

Boolean Options

A boolean options toggles on or off (for example `--interactive` puts you in interactive mode where you are prompted for options; `--no-interactive` turns off interactive mode). Putting --no- in front of the long option toggles the option off. Specifying the short option name always sets the opposite of the default value.

You can group short boolean options. For example, you can use `-Ie` to specify interactive (short option `-I`) and echo (short option `-e`).

Operands

Operands are set off by a space or a tab. They can come in any order in the command syntax. You can use -- with no option after it to separate the options from the operands. Any following arguments are treated as operands, even if they begin with a dash (-). For example, in

```
asadmin> create-jvm-options --instance server1 -- -Xmx1500m
```

`-XMx1500m` is treated as an operand, even though it begins with a dash.

Syntax Example

```
asadmin create-instance [--user admin_user] [--password admin_password]
[-H host_name] [--port port_number] [--sysuser sys_user] [--domain
domain_name] [--local=true/false] [--passwordfile file_name] [--secure | -s]
--instanceport instance_port instance_name
```

In this syntax example `-H` is the short option for hostname, `--user` is a long option with *admin_user* as its argument, and *instance_name* is an operand. Optional options are inside brackets.

The following example shows the syntax with real values. Some of the optional options are not used in the example.

```
asadmin create-instance --user admin --password password -H austen
--port 4848 --instanceport 1024 server2
```

Using Singlemode and Multimode

You can run `asadmin` either in singlemode or multimode. In singlemode you run one command at a time from the command prompt. In multimode you can run multiple commands without needing to reenter environment-level information.

If you are using input from a file, and a command fails, in singlemode the program exits. If you are using multimode and a command fails, you return to the `asadmin` prompt.

Singlemode

If you invoke a single command in the command line interface from a command prompt, you are running in singlemode. The command line interface runs the command and exits back to the command prompt. To run the command line interface from a command prompt, go to the directory *install_dir*/appserv/bin and type your command at the command prompt:

```
> asadmin command options arguments
```

For example:

```
> asadmin create-instance --user admin --password password -H austen
--port 4848 --instanceport 1024 server2
```

Multimode

Multimode lets you set your environment at the outset so that you can run multiple commands without needing to re-enter certain environment-level information, such as your server name, port, and your password. One significant advantage of using multimode is that commands can be entered end executed significantly faster, since `asadmin` stays in memory. If these environment variables are set at the operating system level, multimode picks up those settings. The `asadmin` utility uses these settings until you change them.

On Windows, you are automatically in multimode when you run the `asadmin.bat` file.

On UNIX, to start the `asadmin` utility in multimode from the command line, type:

```
> asadmin multimode
```

When you are in multimode, the command prompt changes to `asadmin`. You can then type your commands into the `asadmin` prompt. You do not need to use the utility name. For example:

```
asadmin> create-instance --user admin --password password -H austen
--port 4848 --instanceport 1024 server2
```

Exit multimode by typing exit or quit. You return to the command prompt.

Multiple Multimode

You can also invoke multimode from within a multimode session, by using the command:

```
asadmin> multimode
```

Once you exit the second multimode environment, you return to your original multimode environment.

For example, if you are administering server1 in multimode, and you want to administer server2 to compare the two, you can invoke multimode for server2 from within multimode for server1. Because you do not need to exit your current multimode session, you can retain your environment settings. When you exit the multimode session you're using for server2, you return to your server1 multimode environment.

Using Interactive and Non-Interactive Options

When you use the command line interface, you can use it in interactive or non-interactive mode. If you choose to use interactive mode, you are prompted for the password if you don't specify it. Interactive mode is enabled by default.

You can disable and enable interactive mode by setting the interactive environment variable using the export command. For more information, see the "Environment Variables Used with export Command" table.

You can use the interactive option in singlemode under all circumstances. You can use the interactive option in multimode when you run one command at a time from the command prompt, and when you run in multimode from a file. However, commands in multimode when piped from an input stream and commands invoked from another program cannot run in interactive mode.

Using the Environment Commands

The asadmin utility contains a set of environment variables that you can set using environment commands. In multimode, once you set these variables you do not need to reset your environment until you exit multimode. You can also set these environment variables at the operating system level; if you do, they are automatically picked up when you enter multimode and they persist after you exit from multimode.

Environment variables are name/value pairs that you can set at any time by assignment. Environment variable constitutes of AS_ADMIN_ prefix to the option name in capitals. For example, to set the Administration Server user, you can type:

```
export AS_ADMIN_USER=administrator
```

Where *administrator* is the administrator's username.

This also makes the value for AS_ADMIN_USER available to asadmin commands; for example:

```
asadmin multimode
asadmin> export AS_ADMIN_HOST=austen
```

For as long as you are in this multimode session, the Administration Server hostname is set to austen, unless you reassign it.

You can also set and export values for multiple environment variables in one step, for example:

```
asadmin> export AS_ADMIN_PORT=4848 AS_ADMIN_USER=admin
```

To see your current environment variable settings, use the export command without arguments:

```
asadmin> export
AS_ADMIN_HOST=austen
AS_ADMIN_PORT=4848
AS_ADMIN_USER=admin
```

Use the `unset` command to remove a variable and its value from the environment. For example:

```
asadmin> unset AS_ADMIN_HOST
```

You can override the set value for an environment variable either by resetting the variable, or by setting a different value as part of an `asadmin` command. For example:

```
asadmin> export AS_ADMIN_HOST=dickens
asadmin> show-instance-status --host austen instance-name
```

This example shows the instance status for an instance in the Administration Server host `austen`, because that value overrides the earlier host value of `dickens`.

If you do not use exported variables, you must provide the following options with most commands or use the default values (for a list of default values, see "Long and Short Option Formats, Default Values, and Environment Variable Equivalents" on page 464):

- --host
- --port
- --user
- --password or --passwordfile
- --secure=true (if secure)
- --instance (if needed)

The following table, "Environment Variables Used with export Command," describes some of the environment variables to use with the export command. These variables are the most commonly used, as they specifically deal with setting up the environment. The first column shows the environment variable name, and the second column shows what it is used for, and the default value if none is set. For a complete list of the environment variables, see "Long and Short Option Formats, Default Values, and Environment Variable Equivalents" on page 464

Table A-2 Environment Variables Used with export Command

Environment Variable	Use
AS_ADMIN_HOST	Hostname of the Administration Server. If no value is specified, uses localhost.
AS_ADMIN_PORT	Port number of the Administration Server. If no value is specified, uses 4848.
AS_ADMIN_USER	Username of the user to run the commands.
AS_ADMIN_PASSWORD	Password of the username that runs the commands. The username and password are used to authenticate the user, to verify that the user is allowed to administer the server. It is the same as the authentication that takes place when you access the Administration Server through the Administration interface.
AS_ADMIN_SECURE	`=true` if secure.
AS_ADMIN_INSTANCE	Sets the instance of the Sun ONE Application Server. Any subsequent commands that use the instance name as an argument (but not those that use it as an operand) use this specified instance.

Using the Password File Option

If you do not want to type the passwords on the command line or set environment variables for the passwords, you can create a password file and use it as an option from the command line.

Every command which has the a `password` option also has the `passwordfile` option that you can use instead. The password file contains the following lines:

`AS_ADMIN_PASSWORD=`*value*

`AS_ADMIN_ADMINPASSWORD=`*value*

`AS_ADMIN_USERPASSWORD=`*value*

If you use the passwordfile option, the passwords in the file are exported to the multimode environment, and subsequent commands without the password options specified use these values.

If you specify both a password and a password file option at the command line, the values in the password file are exported to the multimode environment, but the current command uses the password specified in the password option, because the password option takes precedence over the password file.

Running asadmin Locally or Remotely

Usually the asadmin utility sends its commands through the Administration Server. Therefore, you do not need to run asadmin on the system where the Sun ONE Application Server is installed. However, the Administration Server must be running in order for most asadmin commands to work.

Some commands have the option of being run locally, for example, create-instance. If you use the --local=true option with create-instance, you must run it on the machine where the server is installed, but you do not need to have the Administration Server running in order to create the instance.

Some commands must be run locally. For example, start-appserv, which starts the Administration Server and all its instances, cannot be run remotely because the Administration Server is not running until the command starts it.

For more information on the Administration Server, see Chapter 2, "Setting Administration Server Preferences."

The following commands can be run both locally and remotely:

- create-instance
- delete-instance
- list-instances
- start-instance
- stop-instance
- display-license
- version
- stop-domain
- restart-instance

- list-domains

For these commands, you can choose to run a command locally without specifying the local option. By default, if you specify a value for user, password, host, or port in the command syntax the command is treated as a remote command (though you can still specify local values for these options). If you do not specify values for these options, the command is run locally by default.

When a command is executed locally, if there is a domain option it is required by the command (unless there is only one domain).When a command is executed remotely it ignores the domain option if you specify it.

Using Command Line Invocations

You can invoke a command line in many ways, as described in the following topics:

- Using asadmin from the Command Line
- Using asadmin with Input from a File (Script)
- Using asadmin with Standard Input (Pipe)

Using asadmin from the Command Line

The simplest way to use the commands is one at a time, from the command line. You type in the utility, command, its options and arguments. In multimode you type multiple commands without needing to retype the utility name and environment options (if you have set the environment variables). You can run either singlemode or multimode commands interactively (prompted for additional required input, for example, a password) or non-interactively.

For more information on singlemode and multimode, see “Using Singlemode and Multimode” on page 417.

For more information on using commands interactively, see “Using Interactive and Non-Interactive Options” on page 419.

Example from Command Line

```
> asadmin create-instance --user admin --password password --host
austen --port 4848 --instanceport 1024 server2
```

After the command completes, you return to the operating system prompt.

Using asadmin with Input from a File (Script)

You can create a script which includes many `asadmin` commands. With scripts you can process commands in a batch, set up a job to run at specific times, and otherwise simplify and automate your administration tasks.

To call a script that is in a file, use the syntax:

```
> asadmin multimode --file filename
```

The following is an example of a simple script in a file that you can call in this manner:

```
# Create new instance and start it.
export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=mypassword
AS_ADMIN_HOST=austen AS_ADMIN_PORT=4848
create-instance --instanceport 9000 austen3
start-instance austen3
```

This script sets the environment, creates an instance called `austen3` and starts the new instance. Lines that begin with a number sign (#) are treated as comments and ignored.

Using asadmin with Standard Input (Pipe)

You can pipe input into the `asadmin` utility using the following syntax:

```
cat filename | asadmin multimode
```

This syntax may not work on Windows.

Using Escape Characters

Some characters, the colon (:), the asterisk (*), and the backslash (\), cause errors if you use them in the command syntax unless you use escape characters to set them off. The possibilities for using escape characters vary depending upon what platform you use and whether you use singlemode or multimode.

NOTE You do not need to use escape characters for colons in the `get` and `set` commands.

This section contains the following topics:

- Escape Characters on UNIX in Singlemode
- Escape Characters on Windows in Singlemode

- Escape Characters on All Platforms in Singlemode
- Escape Characters on All Platforms in Multimode

Escape Characters on UNIX in Singlemode

On Solaris, you can use either two backslashes (\ \) or double-quotes (“ ”) to escape restricted characters.

Escape with Backslashes (\\)

For example, when creating a JDBC connection pool with a option whose value includes colons, you could use backslashes (example assumes the environment variables have been set for some properties):

```
asadmin create-jdbc-connection-pool  --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=jdbc\\:oracle\\:thin\\:@asperfsol8\\:1521\\:V8i:user=staging_lo
okup_app:password=staging_lookup_app OraclePoollookup
```

Escape with Quotes

To use quotes in the same example as above, you would enclose the value in double quotes (”) and escape the double quotes with the backslash.

```
asadmin create-jdbc-connection-pool  --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=\"jdbc:oracle:thin:@asperfsol8:1521:V8i\":user=staging_lookup_a
pp:password=staging_lookup_app OraclePoollookup
```

You can also use the methods described in “Escape Characters on All Platforms in Singlemode” on page 426.

Escape Characters on Windows in Singlemode

On windows, you can escape using the backslash character. For example, when creating a JDBC connection pool with a option whose value includes colons, you could use backslashes (example assumes the environment variables have been set for some properties):

```
asadmin create-jdbc-connection-pool  --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=jdbc\:oracle\:thin\:@asperfsol8\:1521\:V8i:user=staging_lookup_
app:password=staging_lookup_app OraclePoollookup
```

You can also use the methods described in “Escape Characters on All Platforms in Singlemode” on page 426.

Escape Characters on All Platforms in Singlemode

On any platform, you can use backslashes to escape the character and enclose the value containing the escaped characters in double quotes. For example, when creating a JDBC connection pool with a option whose value includes colons, you could use the escape characters as follows (example assumes the environment variables have been set for some properties):

```
asadmin create-jdbc-connection-pool  --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url="jdbc\:oracle\:thin\:@iasperfsol8\:1521\:V8i":user=staging_look
up_app:password=staging_lookup_app OraclePoollookup
```

Escape Characters on All Platforms in Multimode

In multimode you can use the following syntax, which only requires quotes, not slashes or backslashes:

```
asadmin> create-jdbc-connection-pool  --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url="jdbc:oracle:thin:@asperfsol8:1521:V8i":user=staging_lookup_app
:password=staging_lookup_app OraclePoollookup
```

Using get and set Commands

Use the get and set commands to access and change configuration settings in the Sun ONE Application Server. In most cases, asadmin commands only set the *required* properties. Use the set command to change values for optional properties.

Table A-3 get and set Commands

Command	Arguments	Use
get	(scope) where scope represents an attribute and is a valid name.	Gets the value of the attribute.

Table A-3 get and set Commands

Command	Arguments	Use
set	(scope=*value*) where scope represents an attribute and is a valid name, and *value* is the value you want to set for that attribute.	Sets the value of the attribute.
reconfig	instance-name	After running any command that modifies the configuration files, you need to run reconfig in order for the changes to be applied to the server. For more information on applying changes / reconfiguring the server, see "Applying Changes to an Application Server Instance" on page 79.

You can get or set multiple attribute values in a single command by using spaces between the attributes. For example:

```
set server1.appReloadPollInterval=20
server1.mime.mime1.file=mime.types
```

You can also use the AS_ADMIN_PREFIX environment variable to set a prefix that will be used by subsequent get and set commands. A period (".") is implicitly inserted between the prefix string and the operand in the get and set commands. For example:

```
asadmin>export AS_ADMIN_PREFIX=server1
asadmin>get *
server1.locale = en_US
server1.appReloadPollInterval = 2
server1.name = server1
...
```

Because the get and set commands require period separators, if an item contains periods in its name you must use the escape character backslash (\) before them. The following example shows a server instance name server2.sun.com with the periods preceded by backslashes:

```
get server2\.sun\.com.*
```

If you do not include the backslashes, you get an error message.

get and set Command Examples

The following examples show how to use the get command to get the values of attributes, and how to use the set command to set values.

MDB Container Service Example

If the application server instance is server1, you can get the values of all mdb-container attributes by using the following command in multimode, with your environment set:

```
asadmin> get server1.mdb-container.*
```

The following is an example of the output from this command, showing current values for the attributes:

```
server1.mdb-container.logLevel = null
server1.mdb-container.steadyPoolSize = 10
server1.mdb-container.idleInPoolTimeoutInSeconds = 600
server1.mdb-container.maxPoolSize = 60
server1.mdb-container.monitoringEnabled = false
server1.mdb-container.poolResizeQuantity = 2
```

To get just the value of the MDB container attribute monitoringEnabled, use the following:

```
asadmin> get server1.mdb-container.montioringEnabled
```

To set the value of the monitoringEnabled attribute to true, use the following:

```
asadmin> set server1.mdb-container.montioringEnabled=true
```

JMS Resource Example

For configuring any of the resources, the attribute should look like:

instancename.resource.primary_key_value.attribute_name

For example:

```
asadmin> get server1.jms-resource.myjms.*
```

Gets all the attributes for the JMS destination resource called myjms. For example:

```
server1.jms-resource.myjms.resType = javax.jms.Topic
server1.jms-resource.myjms.enabled = true
server1.jms-resource.myjms.name = myjms
server1.jms-resource.myjms.description = null
```

To get the value for a single attribute, for example resType:

```
asadmin> get server1.jms-resource.myjms.resType
```

To set an attribute, for example description:

```
asadmin> set server1.jms-resource.myjms.description=mydescription
```

This example sets the description to mydescription.

Getting and Setting Multiple Values Examples

You can get and set multiple values with the same command. To set two attributes at the same time, separate the attributes by spaces. For example:

```
set server1.appReloadPollInterval=20
server1.mime.mime1.file=mime.types
```

Also, you can use the environment variable AS_ADMIN_PREFIX to set a prefix to use for a number of get and set commands.

Monitoring Using get and set Commands

You can also use the get and set commands to monitor a running server. A list command also exists for monitoring. You can set an option, monitor, to true or false. If set to true, you monitor the attributes specified. For more information on using the command-line interface to monitor the Sun ONE Application Server, see "Extracting Monitoring Data Using the CLI" on page 132.

Using Help

Help for every asadmin command is available from the command prompt by typing -h or --help. For example, for help on asadmin, type:

```
asadmin --help
```

You see a list of all asadmin commands.

To get a help for a specific asadmin command, type:

```
asadmin command -h
```

or

```
asadmin command --help
```

The help contains a synopsis, a description of the command, syntax information, examples, and a list of related commands.

Please note that if you use -h or --help anywhere in a command, you will get help for the command. The command will not execute.

You can also access the command-line help pages as manpages in a UNIX environment. For an unbundled installation, add *install_dir*/man to your MANPATH environment variable. Once this is done, you can access manpages for Sun ONE Application Server utilities, for example, by typing `man asadmin` at the command prompt.

Viewing Output and Errors

When a command executes successfully, you see a message informing you of what was done. If the command fails, you see an error message.

This section contains the following topics:

- Viewing the Exit Status
- Viewing Usage

Viewing the Exit Status

In addition to an error message, `asadmin` commands always exit with an exit status. The exit status is 0 if the command is successful and 1 if the command fails.

Exit Status on UNIX

You can check for the exit status at the command prompt by typing `echo $?`.

You can also use the exit codes in scripting; for example, the following Korn shell script uses the exit status to indicate whether the `list-instances` command succeeded or failed:

```
#!/bin/ksh
asadmin list-instances
if  [[ $? = 0 ]]
   then
      echo "success"
   else
      echo "error"
fi
```

Exit Status on Windows

On Windows you can check the exit status in your .bat scripts. For example, the following two scripts show a successful script and the output it returns, and an unsuccessful script and the output it returns:

Success Condition

```
myscript.bat
------------
echo off
echo Processing Command
call asadmin list-instances --domain domain1
if not %errorlevel% EQU 0  goto end
echo Command Succesful
goto program-end
:end
echo Command Failed
:program-end
```

Output:

```
Processing Command
admin-server <not running>
server1 <not running>
Command Succesful
```

Error Condition

```
myscript.bat
-------------
echo off
echo Processing Command
call asadmin list-instances
if not %errorlevel% EQU 0  goto end
echo Command Succesful
goto program-end
:end
echo Command Failed
:program-end
```

Output:

```
Processing Command
No default domain.  Need to enter a domain.
Command Failed
```

Viewing Usage

If you type the command without arguments, you get an error message that includes the syntax for the command. For example:

```
asadmin> create-instance
```

```
Invalid number of operands received
```

```
USAGE: create-instance [--user admin_user] [--password
admin_password] [--host localhost] [--port 4848] [--sysuser
sys_user] [--domain domain_name] [--local=false] [--passwordfile
file_name] [secure | -s] --instanceport instanceport instancename
```

Security Considerations

When you run the command line interface from a command line, you must supply your password with all commands. If you are running in multimode, you must supply your password initially when you set up the environment. If you exit multimode, when you start multimode again you must set up your environment again, including your password. You set passwords using the environment commands. For more information see "Using the Environment Commands" on page 419.

You can also set up a password file so you do not need to type your passwords at the command line. For more information, see "Using the Password File Option" on page 421.

Without the authentication information of a valid username and password, the commands will not execute.

The command line interface has the security measures you've set up for your Sun ONE Application Server. For more information regarding security in the Sun ONE Application Server, see the *Sun ONE Application Server Administrator's Security Guide*.

Concurrent Access Considerations

It is possible that more than one person could attempt to configure a server concurrently using the command line interface and/or the Administration interface. If that happens, the second configuration request is queued until the first one completes. If the request is queued for too long, it times out.

For some commands, the changes don't take effect until you use the `reconfig` command. That means more than one person could edit an attribute before the changes are applied to the server. For more information on `reconfig`, see "Applying Changes to an Application Server Instance" on page 79.

Command Reference

This section contains the following topics:

- List of Commands
- List of Dotted Names and Attributes
- Long and Short Option Formats, Default Values, and Environment Variable Equivalents

List of Commands

The following table shows all the `asadmin` commands and their purpose. For more information on a command's syntax and usage, see the online help.

The left column shows the command name, and the right column shows its use.

Table A-4 `asadmin` Commands

Command	Use
`add-resources`	Adds one or more resources of type jdbc, jms, or javamail.
`create-acl`	Creates an ACL (access control list).
`create-authdb`	Creates an authentication database.
`create-auth-realm`	Creates an authentication realm.
`create-custom-resource`	Creates a custom resource.
`create-domain`	Creates a domain.
`create-file-user`	Creates a file realm user in the keyfile.
`create-http-listener`	Creates an HTTP listener.
`create-http-qos`	Creates HTTP quality of service settings for the application server instance or virtual server.
`create-iiop-listener`	Creates an IIOP listener.
`create-instance`	Creates an application server instance.
`create-javamail-resource`	Creates a Java Mail resource.
`create-jdbc-connection-pool`	Creates a JDBC connection pool.
`create-jdbc-resource`	Creates a JDBC resource.
`create-jmsdest`	Creates a JMS (Java Message Service) destination.

Table A-4 `asadmin` Commands

Command	Use
`create-jms-resource`	Creates a JMS resource.
`create-jndi-resource`	Creates a JNDI resource.
`create-jvm-options`	Creates JVM options in java-config or profiler elements.
`create-lifecycle-module`	Creates a lifecycle module.
`create-mime`	Creates a MIME types file.
`create-persistence-resource`	Creates a persistence manager factory resource.
`create-profiler`	Creates a profiler for the JVM.
`create-ssl`	Creates SSL settings for an HTTP listener, IIOP listener, or IIOP service.
`create-virtual-server`	Creates a virtual server.
`delete-acl`	Deletes an ACL.
`delete-authdb`	Deletes an authentication database.
`delete-auth-realm`	Deletes an authentication realm.
`delete-custom-resource`	Deletes a custom resource.
`delete-domain`	Deletes a domain. This command can only be executed locally.
`delete-file-user`	Deletes a file realm user from the keyfile.
`delete-http-listener`	Deletes an HTTP listener.
`delete-http-qos`	Deletes HTTP quality of service settings for the application server instance or virtual server.
`delete-iiop-listener`	Deletes an IIOP listener
`delete-instance`	Deletes an application server instance.
`delete-javamail-resource`	Deletes a Java Mail resource.
`delete-jdbc-connection-pool`	Deletes a JDBC connection pool.
`delete-jdbc-resource`	Deletes a JDBC resource.
`delete-jmsdest`	Deletes a JMS destination.
`delete-jms-resource`	Deletes a JMS resource.
`delete-jndi-resource`	Deletes a JNDI resource.

Table A-4 asadmin Commands

Command	Use
delete-jvm-options	Deletes JVM options in java-config or profiler elements.
delete-lifecycle-module	Deletes a lifecycle module.
delete-mime	Deletes a MIME types file.
delete-persistence-resource	Deletes a persistence manager factory resource.
delete-profiler	Deletes a JVM profiler.
delete-ssl	Deletes SSL settings for an HTTP listener, IIOP listener, or IIOP service.
delete-virtual server	Deletes a virtual server.
deploy	Deploys an EJB, WEB, connector, appclient, or application component to the application server instance.
deploydir	Deploys an EJB, WEB, connector, appclient, or application component that is in the directory to the application server instance.
disable	Disables a deployed component in the application server instance.
display-license	Displays license information. This command can only be executed locally.
enable	Enables (allows to run) a deployed component in the application server instance.
export	Exports the value of an asadmin environment variable so that it can be used by the subsequent asadmin commands.
get	Gets the value of an attribute.
help	Displays help (description, usage, syntax, examples) for a given command, or general help for asadmin.
install-license	Installs the license file. This command can only be executed locally.
jms-ping	Pings the JMS provider to see if it is running.
list	Lists the configurable elements.
list-acls	Lists ACLs for an application server instance.
list-authdbs	Lists authentication databases.

Table A-4 asadmin Commands

Command	Use
`list-auth-realms`	Lists authentication realms.
`list-components`	Lists the deployed components for a server instance.
`list-custom-resources`	Lists custom resources in a server instance
`list-domains`	Lists domains.
`list-file-users`	Lists all the file realm users in a server instance.
`list-file-groups`	Lists all the groups for a specified file realm user. If you do not specify a user, lists all groups for a server instance.
`list-http-listeners`	Lists HTTP listeners for a server instance.
`list-instances`	Lists application server instances in the domain.
`list-iiop-listeners`	Lists IIOP listeners for a server instance.
`list-javamail-resources`	Lists Java Mail resources for a server instance.
`list-jdbc-connection-pools`	Lists JDBC connection pools for a server instance.
`list-jdbc-resources`	Lists JDBC resources for a server instance.
`list-jmsdest`	Lists JMS destinations for a server instance.
`list-jms-resources`	Lists JMS resources for a server instance
`list-jndi-resources`	Lists JNDI resources for a server instance.
`list-lifecycle-modules`	Lists lifecycle modules for a server instance.
`list-mimes`	Lists MIME types files for a server instance.
`list-persistence-resources`	Lists persistence manager factory resources for a server instance.
`list-profilers`	Lists JVM profilers for a server instance.
`list-sub-components`	Lists one or more EJBs or Servlets in a deployed module or in a module of the deployed application.
`list-virtual-servers`	Lists virtual servers for a server instance.
`multimode`	Allows you to execute multiple command while retaining your environment settings and remaining within asadmin.
`reconfig`	Applies change to the server. Most changes do not take effect until they are applied.

Table A-4 asadmin Commands

Command	Use
restart-instance	Restarts the server instance.
set	Sets the value of an attribute.
show-component-status	Shows the status of a deployed component.
show-instance-status	Shows the status of a server instance (that is, whether it is running or not).
shutdown	Shuts down the Administration Server.
start-appserv	Starts the Administration Server and all the server instances. This command can only be executed locally.
start-domain	Starts all instances in the domain. This command can only be executed locally.
start-instance	Starts the server instance.
stop-appserv	Stops the Administration Server and all the server instances. This command can only be executed locally.
stop-domain	Stops all instances in the domain.
stop-instance	Stops the server instance.
undeploy	Removes the deployed component from the server instance.
unset	Unsets the exported environment variables for asadmin.
update-file-user	Updates an existing file realm user.
version	Displays version information for the Sun ONE Application Server.

List of Dotted Names and Attributes

When you use the get and set commands to get and set attributes, you need to know the names asadmin uses for the services, resources, and so forth so you can use the name to get the attributes for that particular object.

Because the syntax for using these names involves separating names between periods, these names are called dotted names.

Dotted Names Used in asadmin

The following tables list the names used to configure items using the `asadmin`. They are broken out into the following categories:

- Service Names
- Resource Names
- Application Names
- Other Names

Service Names

The following table shows the service names to use to get and set attributes for the services:

Table A-5 Service Names for Command-Line Interface

Service	Dotted Name
JMS service configuration	jms-service
Transaction service configuration	transaction-service
MDB container configuration	mdb-container
EJB container configuration	ejb-container
Web container configuration	web-container
JVM configuration	java-config
ORB configuration	orb or iiop-service
ORB listener configuration	orblistener or iiop-listener
	Note that orblistener or iiop-listener are not valid names as is; both require the names of the listener to follow. For example: `ORB listener configuration orblistener.<listener name> or iiop-listener.<listener name>`
Log configuration	log-service
Security configuration	security-service
HTTP configuration	http-service

Resource Names

The following table shows resource names to use to get and set attributes for the resources. Note that these names are not valid by themselves; they require the name of a resource to follow the resource name.

Table A-6 Resource Names for Command-Line Interface

Resource	Dotted Name
JDBC resource configuration	jdbc-resource
JNDI resource configuration	jndi-resource
JDBC connection pool resource configuration	jdbc-connection-pool
Custom resource configuration	custom-resource
JMS resource configuration	jms-resource
Persistence manager factory resource configuration	persistence-manager-factory-resource
Java mail resource configuration	mail-resource

Application Names

The following table shows the dotted names to use to get and set attributes application-related configuration. Note that these names are not valid by themselves; they require the name of the application to follow.

Table A-7 Application Names for Command-Line Interface

Application Component	Dotted Name
Application configuration	application
EJB module configuration	ejb-module
Web module configuration	web-module
Connector module configuration	connector-module

Other Names

The following table shows the dotted names of other items you can configure using `get` and `set`. Note that these names are not valid by themselves; they require the name of the application to follow; for example, `http-listener.`*listener_name*, `lifecycle-module.`*module-name*, etc.

Table A-8 Other Item Names for Command-Line Interface

Item	Dotted Name
HTTP listener	http-listener or http-server.http-listener
MIME types file	mime
ACL	acl
Virtual server	virtual-server
Authentication databases	auth-db
Security realms	authrealm
Lifecycle module	lifecycle-module
Profiler configuration	profiler
Server configuration	server configuration (name of server instance)

Attributes

The following sections show the attributes for each named item listed above, and provide usage examples. Note that some attributes are read-only (that is, they can only be used with the `get` command, not the `set` command).

NOTE The examples in this section assume the user, password, host and port are defined in the environment variables, and don't list those options in the syntax.

jms-service

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-9 JMS Service Attributes

server.xml Name	asadmin Name
port	port
admin-username	adminUserName
admin-password	adminPassword
log-level	logLevel
enabled	enabled
init-timeout-in-seconds	initTimeoutInSeconds
start-args	startArgs

To get all the attributes from an instance (server1):

```
asadmin> get server1.jms-service.*
```

To get an attribute called `adminPassword`:

```
asadmin> get server1.jms-service.adminPassword
```

To set an attribute called `adminPassword` to a value of `admin`:

```
asadmin> set server1.jms-service.adminPassword=admin
```

transaction-service

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-10 Transaction Service Attributes

server.xml Name	asadmin Name
automatic-recovery	automaticTransactionRecovery
timeout-in-seconds	transactionRecoveryTimeout
tx-log-dir	transactionLogFile
heuristic-decision	heuristicDecision
keypoint-interval	keypointInterval
log-level	logLevel
monitoring-enabled	monitoringEnabled

To get all the attributes from an instance (server1):

```
asadmin> get server1.transaction-service.*
```

To get an attribute called `transactionRecoveryTimeout`:

```
asadmin> get server1.transaction-service.transactionRecoveryTimeout
```

To set an attribute called `transactionRecoveryTimeout` to a value of 49:

```
asadmin> set
server1.transaction-service.transactionRecoveryTimeout=49
```

mdb-container

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-11 MDB Container Attributes

server.xml Name	asadmin Name
steady-pool-size	steadyPoolSize
pool-resize-quantity	poolResizeQuantity
max-pool-size	maxPoolSize
idle-timeout-in-seconds	idleInPoolTimeoutInSeconds
log-level	logLevel
monitoring-enabled	monitoringEnabled

To get all the attributes from an instance (server1):

```
asadmin> get server1.mdb-container.*
```

To get an attribute called `steadyPoolSize`:

```
asadmin> get server1.mdb-container.steadyPoolSize
```

To set an attribute called `steadyPoolSize` to a value of 10:

```
asadmin> set server1.mdb-container.steadyPoolSize=10
```

ejb-container

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-12 EJB Container Attributes

server.xml Name	asadmin Name
steady-pool-size	steadyPoolSize
pool-resize-quantity	poolResizeQuantity
max-pool-size	maxPoolSize
cache-resize-quantity	cacheResizeQuantity
max-cache-size	maxCacheSize
pool-idle-timeout-in-seconds	idleInPoolTimeoutInSeconds
cache-idle-timeout-in-seconds	idleInCacheTimeoutInSeconds
removal-timeout-in-seconds	removalTimeoutInSeconds
victim-selection-policy	victimSelectionPolicy
commit-option	commitOption
log-level	logLevel
monitoring-enabled	monitoringEnabled

To get all the attributes from an instance (server1):

```
asadmin> get server1.ejb-container.*
```

To get an attribute called `maxPoolSize`:

```
asadmin> get server1.ejb-container.maxPoolSize
```

To set an attribute called `maxPoolSize` to a value of 12:

```
asadmin> set server1.ejb-container.maxPoolSize=12
```

web-container

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-13 Web Container Attributes

server.xml Name	asadmin Name
log-level	logLevel
monitoring-enabled	monitoringEnabled (not used)

To get all the attributes from an instance (server1):

```
asadmin> get server1.web-container.*
```

To get an attribute called `logLevel`:

```
asadmin> get server1.web-container.logLevel
```

To set an attribute called `monitoringEnabled` to be WARNING:

```
asadmin> set server1.web-container.logLevel=WARNING
```

java-config

The following table shows the `sserver.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-14 JVM Attributes

server.xml Name	asadmin Name
java-home	javahome
debug-enabled	debugEnabled
debug-options	debugOptions
javac-options	javacoptions
rmic-options	rmicoptions
classpath-prefix	classpathprefix
server-classpath	serverClasspath
classpath-suffix	classpathsuffix
native-library-path-prefix	libpathprefix
native-library-path-suffix	libpathsuffix
env-classpath-ignored	envpathignore

To get all the attributes from an instance (server1):

```
asadmin> get server1.java-config.*
```

To get an attribute called `classpathprefix`:

```
asadmin> get server1.java-config.classpathprefix
```

To set an attribute called `classpathprefix` to a value of `com.sun`:

```
asadmin> set server1.java-config.classpathprefix=com.sun
```

orb or iiop-service

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-15 ORB/IIOP Service Attributes

server.xml Name	asadmin Name
message-fragment-size	msgSize
steady-thread-pool-size	minThreads
max-thread-pool-size	maxThreads
max-connections	maxConnections
idle-thread-timeout-in-seconds	idleThreadTimeout
log-level	log
monitoring-enabled	monitor
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

To get all the attributes from an instance (server1):

```
asadmin> get server1.orb.*
```

or

```
asadmin> get server1.iiop-service.*
```

To get an attribute called `msgSize`:

```
asadmin> get server1.orb.msgSize
```

or

```
asadmin> get server1.iiop-service.msgSize
```

To set an attribute called `idleThreadTimeout` to 300:

```
asadmin> set server1.orb.idleThreadTimeout=300
```

or

```
asadmin> set server1.iiop-service.idleThreadTimeout=300
```

orblistener or iiop-listener

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-16 IIOP Listener Attributes

server.xml Name	asadmin Name
id	id
address	address
port	port
enabled	enabled
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

To get all the attributes from an instance (server1):

```
asadmin> get server1.orblistener.orb_listener_id.*
```

or

```
asadmin> get server1.iiop-listener.orb_listener_id.*
```

To get an attribute called `port`:

```
asadmin> get server1.orblistener.orb_listener_id.port
```

or

```
asadmin> get server1.iiop-listener.orb_listener_id.port
```

To set an attribute called `address` to `bluestar`:

```
asadmin> set server1.orblistener.orb_listener_id.address=bluestar
```

or

```
asadmin> set server1.iiop-listener.orb_listener_id.address=bluestar
```

log-service

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-17 Log Configuration Attributes

server.xml Name	asadmin Name
file	file
level	level
log-stdout	stdout
log-stderr	stderr
echo-log-messages-to-stderr	echoToStderr
create-console	createConsole
log-virtual-server-id	LogVirtualServerId
use-system-logging	useSystemLogging

To get all the attributes from an instance (server1):

```
asadmin> get server1.log-service.*
```

To get an attribute called `level`:

```
asadmin> get server1.log-service.level
```

To set an attribute called `echoToStderr` to true:

```
asadmin> set server1.log-service.echoToStderr=true
```

security-service

The following table shows the server.xml name for the attribute in the left column, and the name used by asadmin in the right column.

Table A-18 Security Realm Configuration Attributes

server.xml Name	asadmin Name
default-realm	defaultRealm
default-principal	defaultPrinicpal
default-principal-password	defaultPrinicpalPassword
anonymous-role	anonymousRole
audit-enabled	auditEnabled
log-level	logLevel

To get all the attributes from an instance (server1):

```
asadmin> get server1.security-service.*
```

To get an attribute called anonymousRole:

```
asadmin> get server1.security-service.anonymousRole
```

To set an attribute called encryptPasswords to true:

```
asadmin> set server1.security-service.auditEnabled=true
```

http-service

The following table shows the server.xml name for the attribute in the left column, and the name used by asadmin in the right column.

Table A-19 HTTP Service Attributes

server.xml Name	asadmin Name
qos-metrics-interval-in-seconds	qos-metrics-interval-in-seconds
qos-recompute-time-interval-in-millis	qos-recompute-time-interval-in-millis
qos-enabled	qos-enabled
bandwidth-limit	bandwidthLimit
enforce-bandwidth-limit	enforceBandwidthLimit

Table A-19 HTTP Service Attributes

server.xml Name	asadmin Name
connection-limit	connectionLimit
enforce-connection-limit	enforceConnectionLimit

To get all the attributes from an instance (server1):

```
asadmin> get server1.http-service.*
```

To get an attribute called `bandwidthLimit`:

```
asadmin> get server1.http-service.bandwidthLimit
```

To set an attribute called `qos-enabled` to true:

```
asadmin> set server1.http-service.qos-enabled=true
```

jdbc-resource

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-20 JDBC Resource Attributes

server.xml Name	asadmin Name
jndi-name	name
pool-name	pool
enabled	enabled
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.jdbc-resource.jdbc_resource_name.*
```

To get an attribute called `pool`:

```
asadmin> get server1.jdbc-resource.jdbc_resource_name.pool
```

To set an attribute called `enabled` to true:

```
asadmin> set server1.jdbc-resource.jdbc_resource_name.enabled=true
```

jndi-resource

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-21 JNDI Resource Attributes

server.xml Name	asadmin Name
jndi-name	name
jndi-lookup-name	LookupName
res-type	resType
factory-class	factory
enabled	enabled
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.jndi-resource.jndi_name.*
```

To get an attribute called `factory`:

```
asadmin> get server1.jndi-resource.jndi_name.factory
```

To set an attribute called `factory` to `com.sun`:

```
asadmin> set server1.jndi-resource.jndi_name.factory=com.sun
```

jdbc-connection-pool

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-22 JDBC Connection Pool Attributes

server.xml Name	asadmin Name
name	name
datasource-classname	dsClassName
res-type	resType
description	description
steady-pool-size	steadyPoolSize

Table A-22 JDBC Connection Pool Attributes

server.xml Name	asadmin Name
max-pool-size	maxPoolSize
max-wait-time-in-millis	maxWaitTime
pool-resize-quantity	resizeValue
idle-timeout-in-seconds	idleTimeout
transaction-isolation-level	transactionIsolationLevel
is-isolation-level-guaranteed	isIsolationLevelGuaranteed
connection-validation-method	validationMethod
is-connection-validation-required	isValidationRequired
fail-all-connections	failAll
validation-table-name	validationTable

To get all the attributes from an instance (server1):

```
asadmin> get server1.jdbc-connection-pool.pool_name.*
```

To get an attribute called `dsClassName`:

```
asadmin> get server1.jdbc-connection-pool.pool_name.dsClassName
```

To set an attribute called `resizeValue` to 2:

```
asadmin> set server1.jdbc-connection-pool.pool_name.resizeValue=2
```

custom-resource

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-23 Custom Resource Attributes

server.xml Name	asadmin Name
jndi-name	name
res-type	resType
factory-class	factory
enabled	enabled
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.custom-resource.jndi_name.*
```

To get an attribute called `factory`:

```
asadmin> get server1.custom-resource.jndi_name.factory
```

To set an attribute called `factory`:

```
asadmin> set server1.custom-resource.jndi_name.factory=myclass
```

jms-resource

The following table shows the `sserver.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-24 JMS Resource Attributes

server.xml Name	asadmin Name
jndi-name	name
res-type	resType
enabled	enabled
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.jms-resource.jms_resource_name.*
```

To get an attribute called `res-type`:

```
asadmin> get server1.jms-resource.jms_resource_name.resType
```

To set an attribute called `enabled` to true:

```
asadmin> set server1.jms-resource.jms_resource_name.enabled=true
```

persistence-manager-factory-resource

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-25 Persistence Manager Factory Resource Attributes

server.xml Name	asadmin Name
jndi-name	jndiName
jdbc-resource-jndi-name	JdbcResourceJndiName
factory-class	factoryClass
enabled	enabled
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.persistence-manager-factory-resource.jndi_name
```

To get an attribute called `factoryClass`:

```
asadmin> get
server1.persistence-manager-factory-resource.jndi_name.factoryClass
```

To set an attribute called `enabled` to true:

```
asadmin> set
server1.persistence-manager-factory-resource.jndi_name.enabled=true
```

mail-resource

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-26 Java Mail Resource Attributes

server.xml Name	asadmin Name
jndi-name	name
enabled	enabled
store-protocol	storeProtocol
store-protocol-class	storeProtocolClass
transport-protocol	transportProtocol
transport-protocol-class	transportProtocolClass
host	host
user	user

Table A-26 Java Mail Resource Attributes

server.xml Name	asadmin Name
from	from
debug	debug
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.mail-resource.jndi_name.*
```

To get an attribute called `host`:

```
asadmin> get server1.mail-resource.jndi_name.host
```

To set an attribute called `enabled` to true:

```
asadmin> set server1.mail-resource.jndi_name.enabled=true
```

application

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-27 Application Attributes

server.xml Name	asadmin Name
name	name
location	location
virtual-servers	virtualServers
description	description
enabled	enabled

To get all the attributes from an instance (server1):

```
asadmin> get server1.application.application_name.*
```

To get an attribute called `location` in an application:

```
asadmin> get server1.application.application_name.location
```

To set an attribute called `location`:

```
asadmin> set server1.application.application_name.location=
"/export/home/as7se/as1/repository/applications/ASConverter"
```

ejb-module

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-28 EJB Module Attributes

server.xml Name	asadmin Name
name	name
location	location
description	description
enabled	enabled

To get all the attributes of stand-alone EJB module in an instance (server1):

```
asadmin> get server1.ejb-module.ejb_jar_name.*
```

To get all the attributes of an EJB module in an application for an instance (server1):

```
asadmin> get
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.*
```

or

```
asadmin>get server1.application.application_name.ejb-module.ejb_jar_name.*
```

To get an attribute called `location` from a stand-alone EJB module:

```
asadmin> get server1.ejb-module.ejb_jar_name.location
```

To get an attribute called `location` from an EJB module in an application:

```
asadmin> get
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.
location
```

or

```
asadmin> get
server1.application.application_name.ejb-module.ejb_jar_name.location
```

To set an attribute called `location` in the stand-alone EJB module:

```
asadmin> set
server1.ejb-module.ejb_jar_name.location="/export/home/as7se/as1/repos
itory/modules/ejb_jar_name"
```

To set an attribute called `location` in the EJB module bundled into an application:

```
asadmin> set
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.
location="/export/home/as7se/as1/repository/modules/ejb_jar_name"
```

or

```
asadmin>set
server1.application.application_name.ejb-module.ejb_jar_name.location="/ex
port/home/as7se/as1/repository/modules/ejb_jar_name"
```

web-module

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-29 WEB Module Attributes

server.xml Name	asadmin Name
name	name
location	location
context-root	contextRoot
virtual-servers	virtualServers
description	description
enabled	enabled

To get all the attributes of stand-alone web module in an instance (server1):

```
asadmin> get server1.web-module.web_war_name.*
```

To get all the attributes of a web module in an application for an instance (server1):

```
asadmin> get server1.web-module.application_name.web_war_name.*
```

To get an attribute called `location` from stand-alone web module:

```
asadmin> get server1.web-module.web_war_name.location
```

To get an attribute called `location` from the web module in an application:

```
asadmin> get server1.web-module.application_name.web_war_name.location
```

To set an attribute called `location` in the standalone web module:

```
asadmin> set server1.web-module.war-ic.location=
"/export/home/as7se/as1/repository/modules/web_war_name"
```

To set an attribute called `location` in the web module bundled into an application:

```
asadmin> set server1.web-module.application_name.web_war_name.location=
"/export/home/as7se/as1/repository/modules/web_war_name"
```

connector-module

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-30 Connector Module Attributes

server.xml Name	asadmin Name
name	name
location	location
description	description
enabled	enabled

To get all the attributes of standalone connector module in an instance (server1):

```
asadmin> get server1.connector-module.connector_rar_name.*
```

To get an attribute called `location` from standalone connector module:

```
asadmin> get server1.connector-module.connector_rar_name.location
```

To set an attribute called `location` in the standalone connector module:

```
asadmin> set server1.connector-module.connector_rar_name.location=
"/export/home/as7se/as1/repository/modules/connector_rar_name"
```

http-listener or http-server.http-listener

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-31 HTTP Listener Attributes

server.xml Name	asadmin Name
id	id

Table A-31 HTTP Listener Attributes

server.xml Name	asadmin Name
address	address
port	port
family	family
acceptor-threads	acceptorThreads
blocking-enabled	blockingEnabled
security-enabled	securityEnabled
default-virtual-server	defaultVirtualServer
server-name	serverName
enabled	enabled
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

To get all the attributes from an instance (server1):

```
asadmin> get server1.http-listener.http_listener_name.*
```

or

```
asadmin> get server1.http-server.http-listener.http_listener_name.*
```

To get an attribute called `factory`:

```
asadmin> get server1.httpl-listener.http_listener_name.address
```

or

```
asadmin> get server1.http-server.http-listener.http_listener_name.address
```

To set an attribute called `address` to the IP address `0.0.0.0`:

```
asadmin> set server1.http-listener.http_listener_name.address=0.0.0.0
```

or

```
asadmin> set
server1.http-server.http-listener.http_listener_name.address=0.0.0.0
```

mime

The following table shows the server.xml name for the attribute in the left column, and the name used by asadmin in the right column.

Table A-32 MIME Types Attributes

server.xml Name	asadmin Name
id	id
file	file

To get all the attributes from an instance (server1):

```
asadmin> get server1.mime.mime_name.*
```

To get an attribute called file:

```
asadmin> get server1.mime.mime_name.file
```

To set an attribute called file to mime.types:

```
asadmin> set server1.mime.mime_name.file=mime.types
```

acl

The following table shows the server.xml name for the attribute in the left column, and the name used by asadmin in the right column.

Table A-33 ACL Attributes

server.xml Name	asadmin Name
id	id
file	file

To get all the attributes from an instance (server1):

```
asadmin> get server1.acl.acl_name.*
```

To get an attribute called `file`:

```
asadmin> get server1.acl.acl_name.file
```

To set an attribute called `file`:

```
asadmin> set server1.acl.acl_name.file=com/as1.acl
```

virtual-server

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-34 Virtual Server Attributes

server.xml Name	asadmin Name
id	id
http-listeners	httpListeners
config-file	configFile
default-object	defaultObject
accept-language	acceptLanguage
log-file	logFile
default-web-module	defaultWebModule
hosts	hosts
mime	mime
state	state
acls	acls
bandwidth-limit	bandwidthLimit
enforce-bandwidth-limit	enforceBandwidthLimit
connection-limit	connectionLimit
enforce-connection-limit	enforceConnectionLimit
property name="dir" value=	property.dir
property name="nice" value=	property.nice
property name="user" value=	property.user
property name="group" value=	property.group
property name="chroot" value=	property.chroot

Table A-34 Virtual Server Attributes

server.xml Name	asadmin Name
property name="docroot" value=	property.docroot
property name="accesslog" value=	property.accesslog

To get all the attributes from an instance (server1):

```
asadmin> get instance_name.virtual-server.vserver_id.*
```

For example:

```
asadmin> get server1.virtual-server.server1.*
```

To get an attribute called `httpListeners` for virtual server server1:

```
asadmin> get server1.virtual-server.server1.httpListeners
```

To set an attribute called `acceptLanguage` to false:

```
asadmin> set server1.virtual-acceptLanguage=false
```

auth-db

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-35 Authentication Database Attributes

server.xml Name	asadmin Name
id	id
database	database
basedn	basedn
certmaps	certmaps

To get all the attributes from an instance:

```
asadmin> get instancename.virtual-server.vserver_id.auth-db.authdb_id.*
```

For example, for the instance server1, virtual server server1:

```
asadmin> get server1.virtual-server.server1.auth-db.authdb_id.*
```

To get an attribute called `database`:

```
asadmin> get server1.virtual-server.server1.auth-db.authdb_id.database
```

To set an attribute called `database`:

```
asadmin> set
server1.virtual-server.server1.auth-db.authdb_id.database=Oracle
```

authrealm

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-36 Authorization Realm Attributes

server.xml Name	asadmin Name
name	name
classname	classname

To get all the attributes from an instance (server1):

```
asadmin> get server1.authrealm.authrealm_id.*
```

To get an attribute called `classname`:

```
asadmin> get server1.authrealm.authrealm_id.classname
```

To set an attribute called `classname`:

```
asadmin> set
server1.authrealm.authrealm_id.classname=com.sun.as.security.auth.real
m.sharedpassword.SharedPasswordRealm
```

lifecycle-module

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-37 LifeCycle Module Attributes

server.xml Name	asadmin Name
name	name
enabled	enabled
class-name	className
classpath	classPath

Table A-37 LifeCycle Module Attributes

server.xml Name	asadmin Name
load-order	loadOrder
is-failure-fatal	isFailureFatal
description	description

To get all the attributes from an instance (server1):

```
asadmin> get server1.lifecycle-module.lifecycle_module_id.*
```

To get an attribute called `className` for a lifecycle module:

```
asadmin> get server1.lifecycle-module.lifecycle_module_id.className
```

To set an attribute called `className`:

```
asadmin> set
server1.lifecycle-module.lifecycle_module_id.className=com.lifecycle_module_id.
lifecycle
```

profiler

The following table shows the `server.xml` name for the attribute in the left column, and the name used by `asadmin` in the right column.

Table A-38 JVM Profiler Configuration Attributes

server.xml Name	asadmin Name
name	name
classpath	classPath
native-library-path	nativeLibraryPath
enabled	enabled

To get all the attributes from an instance (server1):

```
asadmin> get server1.profiler.*
```

To get an attribute called `enabled`:

```
asadmin> get server1.profiler.enabled
```

To set an attribute called `enabled` to false

```
asadmin> set server1.profiler.enabled=false
```

server configuration (name of server instance)

The following table shows the server.xml name for the attribute in the left column, and the name used by asadmin in the right column.

Table A-39 Server Configuration Attributes

server.xml Name	asadmin Name
instance-name	name
locale	locale
log-root	logRoot
session-store	sessionStore
application-root	applicationRoot
dynamic-reload-enabled	appDynamicReloadEnabled
dynamic-reload-poll-interval-in-seconds	appReloadPollInterval

To get all the attributes from an instance (server1):

```
asadmin> get server1.*
```

To get an attribute called logRoot:

```
asadmin> get server1.logRoot
```

To set an attribute called logRoot:

```
asadmin> set server1.logRoot="/space/log"
```

Long and Short Option Formats, Default Values, and Environment Variable Equivalents

The following table lists the long and short formats for command line options. If there is no short format listed, a short format of the option is not available.

Table A-40 Short and Long Options, Default Values, and Environment Variables

Option Name	Long Format	Short Format	Default Value	Environment Variable
acceptlang	--acceptlang			AS_ADMIN_ACCEPT_
acceptorthreads	--acceptorthreads			AS_ADMIN_ACCEPTOR_THREADS
acls	--acls			AS_ADMIN_ACLS
address	--address			AS_ADMIN_ADDRESS
adminpassword	--adminpassword			AS_ADMIN_ADMINPASSWD
adminport	--adminport		4848	AS_ADMIN_ADMINPORT
adminuser	--adminuser			AS_ADMIN_ADMINUSER
basedn	--basedn			AS_ADMIN_BASEDN
blockingenabled	--blockingenabled			AS_ADMIN_BLOCKINGENABLED
bwlimit	--bwlimit			AS_ADMIN_BWLIMIT
certmaps	--certmaps			AS_ADMIN_CERTMAPS
certname	--certname			AS_ADMIN_CERTNAME
classname	--classname			AS_ADMIN_CLASSNAME
classpath	--classpath			AS_ADMIN_CLASSPATH
clientauthenabled	--clientauthenabled			AS_ADMIN_CLIENTAUTHENABLED
configfile	--configfile			AS_ADMIN_CONFIGFILE
connectionpoolid	--connectionpoolid			AS_ADMIN_CONNECTIONPOOLID
connlimit	--connlimit			AS_ADMIN_CONNLIMIT
contextroot	--contextroot			AS_ADMIN_CONTEXTROOT
database	--database			AS_ADMIN_DATABASE
debug	--debug		false	AS_ADMIN_DEBUG
defaultobj	--defaultobj			AS_ADMIN_DEFAULTOBJ
defaultwebmodule	--defaultwebmodule			AS_ADMIN_DEFAULTWEBMODULE
description	--description			AS_ADMIN_DESCRIPTION

Table A-40 Short and Long Options, Default Values, and Environment Variables

Option Name	Long Format	Short Format	Default Value	Environment Variable
discardmanualchanges	--discardmanualchanges	-d	false	AS_ADMIN_DISCARDMANUALCHANGES
echo	--echo	-e	false	AS_ADMIN_ECHO
enabled	--enabled			AS_ADMIN_ENABLED
enforcebwlimit	--enforcebwlimit			AS_ADMIN_ENFORCEBWLIMIT
enforceconnlimit	--enforceconnlimit			AS_ADMIN_ENFORCECONNLIMIT
failconnection	--failconnection		false	AS_ADMIN_FAILCONNECTION
failurefatal	--failurefatal		false	AS_ADMIN_FAILUREFATAL
family	--family			AS_ADMIN_FAMILY
file	--file	-f		AS_ADMIN_FILE
force	--force	-F	true	AS_ADMIN_FORCE
help	--help	-h		AS_ADMIN_HELP
host	--host	-H		AS_ADMIN_HOST
hosts	--hosts			AS_ADMIN_HOSTS
httplistenerid	--httplistenerid			AS_ADMIN_HTTPLISTENERID
httplisteners	--httplisteners			AS_HTTP_LISTENERS
idletimeout	--idletimeout		300	AS_ADMIN_IDLETIMEOUT
instance	--instance	-i	server1	AS_ADMIN_INSTANCE
instanceport	--instanceport			AS_ADMIN_INSTANCEPORT
interactive	--interactive	-I	true	AS_AMDIN_INTERACTIVE
isconnectvalidaterequired	--isconnectvalidaterequired		false	AS_ADMIN_ISCONNECTVALIDATEREQUIRED
jdbcjndiname	--jdbcjndiname	-a		AS_ADMIN_JDBCJNDINAME
jndilookupname	--jndilookupname	-l		AS_ADMIN_JNDILOOKUPNAME
keepmanualchanges	--keepmanualchanges	-k	false	AS_ADMIN_KEEPMANUALCHANGES
loadorder	--loadorder			AS_ADMIN_LOADORDER
local	--local	-l	false	

Table A-40 Short and Long Options, Default Values, and Environment Variables

Option Name	Long Format	Short Format	Default Value	Environment Variable
logfile	--logfile			AS_ADMIN_LOGFILE
maxpoolsize	--maxpoolsize		32	AS_ADMIN_MAXPOOLSIZE
maxwait	--maxwait		6000	AS_ADMIN_MAXWAIT
mime	--mime			AS_ADMIN_MIME
mimefile	--mimefile			AS_ADMIN_MIMEFILE
monitor	--monitor	-m	false	AS_ADMIN_MONITOR
name	--name	-n		AS_ADMIN_NAME
nativelibpath	--nativelibpath			AS_ADMIN_NATIVELIBPATH
objtype	--objtype	-o		AS_ADMIN_OBJTYPE
password	--password	-w		AS_ADMIN_PASSWORD
poolresize	--poolresize		2	AS_ADMIN_POOLRESIZE
port	--port	-p	8000	AS_ADMIN_PORT
prefix	--prefix	-x		AS_ADMIN_PREFIX
printprompt	--printprompt	-P	true	AS_ADMIN_PROMPT
property	--property			AS_ADMIN_PROPERTY
securityenabled	--securityenabled			AS_ADMIN_SECURITYENABLED
servername	--servername			AS_ADMIN_SERVERNAME
ssl2ciphers	--ssl2ciphers			AS_ADMIN_SSL2CIPHERS
ssl2enabled	--ssl2enabled			AS_ADMIN_SSL2ENABLED
ssl3enabled	--ssl3enabled			AS_ADMIN_SSL3ENABLED
ssl3tlsciphers	--ssl3tlsciphers			AS_ADMIN_SSL3TLSCIPHERS
state	--state			AS_ADMIN_STATE
steadypoolsize	--steadypoolsize	8		AS_ADMIN_STEADYPOOLSIZE
storeprotocol	--storeprotocol			AS_ADMIN_STOREPROTOCOL
storeprotocolclass	--storeprotocolclass			AS_ADMIN_STOREPROTOCOLCLASS
tlsenabled	--tlsenabled			AS_ADMIN_TLSENABLED
tlsrollbackenabled	--tlsrollbackenabled			AS_ADMIN_TLSROLLBACKENABLED

Table A-40 Short and Long Options, Default Values, and Environment Variables

Option Name	Long Format	Short Format	Default Value	Environment Variable
transprotocol	--transprotocol		smtp	AS_ADMIN_TRANSPROTOCOL
type	--type			S_ADMIN_TRANSPROTOCOLCLASS
upload	--upload	-U	true	AS_ADMIN_TYPE
url	--url			AS_ADMIN_URL
user	--user	-u		AS_ADMIN_USER
validationmethod	--validationmethod		auto-commit	AS_ADMIN_VALIDATIONMETHOD
validationtable	--validationtable			AS_ADMIN_VALIDATIONTABLE
version	--version	-v		AS_AMDIN_VERSION
virtualserver	--virtualserver			AS_ADMIN_VIRTUALSERVER

Appendix B

Third Party Copyright Notices

This product includes code licensed from RSA Security, Inc.

Portions of this product were developed using ANTLR. ANTLR 1989-2000 developed by jGuru.com, http://www.ANTLR.org and http://www.jGuru.com.

This product includes software developed through the Netbeans Project at http://www.netbeans.org under the Sun Public License. Such software, if available, may be found at www.netbeans.org.

This product includes Perl. A copy of Perl, if available, may be found at http://public.ActiveState.com/gsar/APC/.

This product includes software developed through the Exolab Project (http://www.exolab.org).

This product includes software developed through the DOM4J Project (http://dom4j.org/).

This product includes software developed by Apache Foundation. Copyright (c) 1999-2001 The Apache Software Foundation. All rights reserved.

This product includes software developed by The Regents of University of California. Copyright (c) 1991, 1993 The Regents of University of California. All rights reserved.

This product includes software developed by International Business Machines Corporation. Copyright (c) 1995-2001 International Business Machines Corporation and others. All rights reserved. The IBM code was obtained under the ICU License. See below.

ICU License - ICU 1.8.1 and later.

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2001 International Business Machines Corporation and others All rights reserved.

Glossary

This glossary provides definitions for common terms used to describe the Sun ONE Application Server deployment and development environment. For a glossary of standard J2EE terms, please see the J2EE glossary at:

```
http://java.sun.com/j2ee/glossary.html
```

access control The means of securing your Sun ONE Application Server by controlling who and what has access to it.

ACL Access Control List. ACLs are text files that contain lists identifying who can access the resources stored on your Sun ONE Application Server. *See also* general ACL.

activation The process of transferring an enterprise bean's state from secondary storage to memory.

Administration interface The set of browser based forms used to configure and administer the Sun ONE Application Server. S*ee also* CLI.

administration server An application server instance dedicated to providing the administrative functions of the Sun ONE Application Server, including deployment, browser-based administration, and access from the command-line interface (CLI) and Integrated Development Environment (IDE).

administrative domain Multiple administrative domains is a feature within the Sun ONE Application Server that allows different administrative users to create and manage their own domains. A domain is a set of instances, created using a common set of installed binaries in a single system.

API Applications Program Interface. A set of instructions that a computer program can use to communicate with other software or hardware that is designed to interpret that API.

applet A small application written in Java that runs in a web browser. Typically, applets are called by or embedded in web pages to provide special functionality. By contrast, a *servlet* is a small application that runs on a server.

application A group of components packaged into an .ear file with a J2EE application deployment descriptor. *See also* component, module.

application client container *See* container.

application server A reliable, secure, and scalable software platform in which business applications are run. Application servers typically provide high-level services to applications, such as component lifecycle, location, and distribution and transactional resource access,

application tier A conceptual division of a J2EE application. *Client tier*: the user interface (UI). End users interact with client software (such as a web browser) to use the application. *Server tier*: the business logic and presentation logic that make up your application, defined in the application's components. *Data tier*: the data access logic that enables your application to interact with a data source.

assembly The process of combining discrete components of an application into a single unit that can be deployed. S*ee also* deployment.

asynchronous communication A mode of communication in which the sender of a message need not wait for the sending method to return before it continues with other work.

attribute A name-value pair in a request object that can be set by a servlet. Also a name-value pair that modifies an element in an XML file. Contrast with *parameter*. More generally, an attribute is a unit of metadata.

auditing The method(s) by which significant events are recorded for subsequent examination, typically in error or security breach situations.

authentication The process by which an entity (such as a user) proves to another entity (such as an application) that it is acting on behalf of a specific identity (the user's security identity). Sun ONE Application Server supports basic, form-based, and SSL mutual authentication. *See also* client authentication, digest authentication, host-IP authentication, pluggable authentication.

authorization The process by which access to a method or resource is determined. Authorization in the J2EE platform depends upon whether the user associated with a request through authentication is in a given security role. For example, a human resources application may authorize managers to view personal employee information for all employees, but allow employees to only view their own personal information.

backup store A repository for data, typically a file system or database. A backup store can be monitored by a background thread (or sweeper thread) to remove unwanted entries.

bean-managed persistence Data transfer between an entity bean's variables and a data store. The data access logic is typically provided by a developer using Java Database Connectivity (JDBC) or other data access technologies. *See also* container-managed persistence.

bean-managed transaction Where transaction demarcation for an enterprise bean is controlled programmatically by the developer. *See also* container-managed transaction.

BLOB Binary Large OBject. A data type used to store and retrieve complex object fields. BLOBs are binary or serializable objects, such as pictures, that translate into large byte arrays, which are then serialized into container-managed persistence fields.

BMP *See* bean-managed persistence.

BMT *See* bean-managed transaction.

broker The Sun ONE Message Queue entity that manages JMS message routing, delivery, persistence, security, and logging, and which provides an interface that allows an administrator to monitor and tune performance and resource use.

business logic The code that implements the essential business rules of an application, rather than data integration or presentation logic.

CA See certificate authority or connector architecture.

cached rowset A `CachedRowSet` object permits you to retrieve data from a data source, then detach from the data source while you examine and modify the data. A cached row set keeps track both of the original data retrieved, and any changes made to the data by your application. If the application attempts to update the original data source, the row set is reconnected to the data source, and only those rows that have changed are merged back into the database.

Cache Control Directives Cache-control directives are a way for Sun ONE Application Server to control what information is cached by a proxy server. Using cache-control directives, you override the default caching of the proxy to protect sensitive information from being cached, and perhaps retrieved later. For these directives to work, the proxy server must comply with HTTP 1.1.

callable statement A class that encapsulates a database procedure or function call for databases that support returning result sets from stored procedures.

certificate Digital data that specifies the name of an individual, company, or other entity, and certifies that the public key included in the certificate belongs to that entity. Both clients and servers can have certificates.

certificate authority A company that sells certificates over the Internet, or a department responsible for issuing certificates for a company's intranet or extranet.

cipher A cryptographic algorithm (a mathematical function), used for encryption or decryption.

CKL Compromised Key List. A list, published by a certificate authority, that indicates any certificates that either client users or server users should no longer trust. In this case, the key has been compromised. *See also* CRL.

classloader A Java component responsible for loading Java classes according to specific rules. *See also* classpath.

classpath A path that identifies directories and JAR files where Java classes are stored. *See also* classloader.

CLI Command-line interface. An interface that enables you to type executable instructions at a user prompt. S*ee also* Administration interface.

client authentication The process of authenticating client certificates by cryptographically verifying the certificate signature and the certificate chain leading to the CA on the trust CA list. See also authentication, certificate authority.

client contract A contract that determines the communication rules between a client and the EJB container, establishes a uniform development model for applications that use enterprise beans, and guarantees greater reuse of beans by standardizing the relationship with the client.

CMP *See* container-managed persistence.

CMR *See* container-managed relationship.

CMT *See* container-managed transaction.

co-locate To position a component in the same memory space as a related component in order avoid remote procedure calls and improve performance.

column A field in a database table.

commit To complete a transaction by sending the required commands to the database. *See* rollback, transaction.

component A web application, enterprise bean, message-driven bean, application client, or connector. *See also* application, module.

component contract A contract that establishes the relationship between an enterprise bean and its container.

configuration The process of tuning the server or providing metadata for a component. Normally, the configuration for a specific component is kept in the component's deployment descriptor file. *See also* administration server, deployment descriptor.

connection factory An object that produces connection objects that enable a J2EE component to access a resource. Used to create JMS connections (TopicConnection or QueueConnection) which allow application code to make use of the provided JMS implementation. Application code uses the JNDI Service to locate connection factory objects using a JNDI Name.

Connection Pool allows highly efficient access to a database by caching and reusing physical connections, thus avoiding connection overhead and allowing a small number of connections to be shared between a large number of threads. *See also* JDBC connection pool

connector A standard extension mechanism for containers to provide connectivity to EISs. A connector is specific to an EIS and consists of a resource adapter and application development tools for EIS connectivity. The resource adapter is plugged in to a container through its support for system level contracts defined in the connector architecture.

connector architecture An architecture for the integration of J2EE applications with EISs. There are two parts to this architecture: a EIS vendor-provided resource adapter and a J2EE server that allows this resource adapter to plug in. This architecture defines a set of contracts that a resource adapter has to support to plug in to a J2EE server, for example, transactions, security and resource management.

container An entity that provides life cycle management, security, deployment, and runtime services to a specific type of J2EE component. Sun ONE Application Server provides web and EJB containers, and supports application client containers. *See also* component.

container-managed persistence Where the EJB container is responsible for entity bean persistence. Data transfer between an entity bean's variables and a data store, where the data access logic is provided by the Sun ONE Application Server. *See also* bean-managed persistence.

container-managed relationship A relationship between fields in a pair of classes where operations on one side of the relationship affect the other side.

container-managed transaction Where transaction demarcation for an enterprise bean is specified declaratively and automatically controlled by the EJB container *See also* bean-managed transaction.

control descriptor A set of enterprise bean configuration entries that enable you to specify optional individual property overrides for bean methods, plus enterprise bean transaction and security properties.

conversational state Where the state of an object changes as the result of repeated interactions with the same client. *See also* persistent state.

cookie A small collection of information that can be transmitted to a calling web browser, then retrieved on each subsequent call from that browser so the server can recognize calls from the same client. Cookies are domain-specific and can take advantage of the same web server security features as other data interchange between your application and the server.

CORBA Common Object Request Broker Architecture. A standard architecture definition for object-oriented distributed computing.

COSNaming Service An an IIOP-based naming service.

CosNaming provider To support a global JNDI name space (accessible to IIOP application clients), Sun ONE Application Server includes J2EE based CosNaming provider which supports binding of CORBA references (remote EJB references).

create method A method for customizing an enterprise bean at creation.

CRL Certificate Revocation List. A list, published by a certificate authority, that indicates any certificates that either client users or server users should no longer trust. In this case, the certificate has been revoked. *See also* CKL.

data access logic Business logic that involves interacting with a data source.

database A generic term for Relational Database Management System (RDBMS). A software package that enables the creation and manipulation of large amounts of related, organized data.

database connection A database connection is a communication link with a database or other data source. Components can create and manipulate several database connections simultaneously to access data.

data source A handle to a source of data, such as a database. Data sources are registered with the iPlanet Application Server and then retrieved programmatically in order to establish connections and interact with the data source. A data source definition specifies how to connect to the source of data.

DataSource Object A DataSource object has a set of properties that identify and describe the real world data source that it represents.

declarative security Declaring security properties in the component's configuration file and allowing the component's container (for instance, a bean's container or a servlet engine) to manage security implicitly. This type of security requires no programmatic control. Opposite of programmatic security. *See* container-managed persistence.

declarative transaction See container-managed transaction.

decryption The process of transforming encrypted information so that it is intelligible again.

delegation An object-oriented technique for using the composition of objects as an implementation strategy. One object, which is responsible for the result of an operation, delegates the implementation to another object, its delegatee. For example, a classloader often delegates the loading of some classes to its parent.

deployment The process of distributing the files required by an application to an application server to make the application available to run on the application server. *See also* assembly.

deployment descriptor An XML file provided with each module and application that describes how they should be deployed. The deployment descriptor directs a deployment tool to deploy a module or application with specific container options and describes specific configuration requirements that a deployer must resolve.

destination resource An objects that represents Topic or Queue destinations. Used by applications to read / write to Queues or publish / subscribe to Topics. Application code uses the JNDI Service to locate JMS resource objects using a JNDI Name.

digest authentication A for of authentication that allows the user to authenticate based on user name and password without sending the user name and password as cleartext.

digital signature an electronic security mechanism used to authenticate both a message and the signer.

directory server See Sun ONE Directory Server.

Distinguished Name *See* DN, DN attribute.

distributable session A user session that is distributable among all servers in a cluster.

distributed transaction A single transaction that can apply to multiple heterogeneous databases that may reside on separate servers.

Document Root The document root (sometimes called the primary document directory) is the central directory that contains all the virtual server's files you want to make available to remote clients.

Domain Registry The Domain Registry is a single data structure that contains domain-specific information, for all the domains created and configured on an installation of Sun ONE Application Server, such as domain name, domain location, domain port, domain host.

DN Distinguished Name. The string representation for the name of an entry in a directory server.

DN attribute Distinguished Name attribute. A text string that contains identifying information for an associated user, group, or object.

DTD Document Type Definition. A description of the structure and properties of a class of XML files.

dynamic redeployment The process of redeploying a component without restarting the server.

dynamic reloading The process of updating and reloading a component without restarting the server. By default, servlet, JavaServer Page (JSP), and enterprise bean components can be dynamically reloaded. Also known as versioning.

EAR file Enterprise ARchive file. An archive file that contains a J2EE application. EAR files have the `.ear` extension. *See also* JAR file.

e-commerce Electronic commerce. A term for business conducted over the Internet.

EIS Enterprise Information System. This can be interpreted as a packaged enterprise application, a transaction system, or a user application. Often referred to as an EIS. Examples of EISs include: R/3, PeopleSoft, Tuxedo, and CICS.

EJB container *See* container.

EJB QL EJB Query Language. A query language that provides for navigation across a network of entity beans defined by container-managed relationships.

EJB technology An enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, remote clients can access the inventory services provided by the application. *See also* container, entity bean, message-driven bean, and session bean.

ejbc utility The compiler for enterprise beans. It checks all EJB classes and interfaces for compliance with the EJB specification, and generates stubs and skeletons.

element A member of a larger set; for example, a data unit within an array, or a logic element. In an XML file, it is the basic structural unit. An XML element contains subelements or data, and may contain attributes.

encapsulate To localize knowledge within a module. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but if the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten.

encryption The process of transforming information so it is unintelligible to anyone but the intended recipient.

entity bean An enterprise bean that relates to physical data, such as a row in a database. Entity beans are long lived, because they are tied to persistent data. Entity beans are always transactional and multi-user aware. *See* message-driven bean, read-only bean, session bean.

ERP Enterprise Resource Planning. A multi-module software system that supports enterprise resource planning. An ERP system typically includes a relational database and applications for managing purchasing, inventory, personnel, customer service, shipping, financial planning, and other important aspects of the business.

event A named action that triggers a response from a module or application.

external JDNI resource Allows the JNDI Service to act as a bridge to a remote JNDI server.

facade Where an application-specific stateful session bean is used to manage various Enterprise JavaBeans (EJBs).

factory class A class that creates persistence managers. *See also* connection factory.

failover A recovery process where a bean can transparently survive a server crash.

finder method Method which enables clients to look up a bean or a collection of beans in a globally available directory.

File Cache The file cache contains information about files and static file content. The file cache is turned on by default.

firewall an electronic boundary that allows a network administrator to restrict the flow of information across networks in order to enforce security.

form action handler A specially defined method in servlet or application logic that performs an action based on a named button on a form.

FQDN Fully Qualified Domain Name. The full name of a system, containing its hostname and its domain name.

general ACL A named list in the Sun ONE Directory Server that relates a user or group with one or more permissions. This list can be defined and accessed arbitrarily to record any set of permissions.

generic servlet A servlet that extends `javax.servlet.GenericServlet`. Generic servlets are protocol-independent, meaning that they contain no inherent support for HTTP or any other transport protocol. Contrast with HTTP servlet.

global database connection A database connection available to multiple components. Requires a resource manager.

global transaction A transaction that is managed and coordinated by a transaction manager and can span multiple databases and processes. The transaction manager typically uses the XA protocol to interact with the database backends. *See* local transaction.

granularity level The approach to dividing an application into pieces. A *high level of granularity* means that the application is divided into many smaller, more narrowly defined Enterprise JavaBeans (EJBs). A *low level of granularity* means the application is divided into fewer pieces, producing a larger program.

group A group of users that are related in some way. Group membership is usually maintained by a local system administrator. *See* user, role.

handle An object that identifies an enterprise bean. A client may serialize the handle, and then later deserialize it to obtain a reference to the bean.

Heuristic Decision The transactional mode used by a particular transaction. A transaction has to either Commit or Rollback.

home interface A mechanism that defines the methods that enable a client to create and remove an enterprise bean.

host-IP authentication A security mechanism used for of limiting access to the Administration Server, or the files and directories on a web site by making them available only to clients using specific computers.

HTML Hypertext Markup Language. A coding markup language used to create documents that can be displayed by web browsers. Each block of text is surrounded by codes that indicate the nature of the text.

HTML page A page coded in HTML and intended for display in a web browser.

HTTP Hypertext Transfer Protocol. The Internet protocol that fetches hypertext objects from remote hosts. It is based on TCP/IP.

HTTP servlet A servlet that extends `javax.servlet.HttpServlet`. These servlets have built-in support for the HTTP protocol. Contrast with generic servlet.

HTTPS HyperText Transmission Protocol, Secure. HTTP for secure transactions.

IDE Integrated Development Environment. Software that allows you to create, assemble, deploy, and debug code from a single, easy-to-use interface.

IIOP Internet Inter-ORB Protocol. Transport-level protocol used by both Remote Method Invocation (RMI) over IIOP and Common Object Request Broker Architecture (CORBA).

IIOP Listener The IIOP listener is a listen socket that listens on a specified port and accepts incoming connections from CORBA based client application

IMAP Internet Message Access Protocol.

IP address A structured, numeric identifier for a computer or other device on a TCP/IP network. The format of an IP address is a 32-bit numeric address written as four numbers separated by periods. Each number can be zero to 255. For example, 123.231.32.2 could be an IP address.

isolation level *See* transaction isolation level.

J2EE Java 2 Enterprise Edition. An environment for developing and deploying multi-tiered, web-based enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing these applications.

JAF The JavaBeans Activation Framework (JAF) integrates support for MIME data types into the Java platform. See Mime Types.

JAR file Java ARchive file. A file used for aggregating many files into one file. JAR files have the.`jar` extension.

JAR file contract Java ARchive contract that specifies what information must be in the enterprise bean package.

JAR file format Java ARchive file format. A platform-independent file format that aggregates many files into one file. Multiple applets and their requisite components (class files, images, sounds, and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction. The JAR files format also supports file compression and digital signatures.

JavaBean A portable, platform-independent reusable component model.

Java IDL Java Interface Definition Language. APIs written in the Java programming language that provide a standards-based compatibility and connectivity with Common Object Request Broker Architecture (CORBA).

JavaMail session An object used by an application to interact with a mail store. Application code uses the JNDI Service to locate JavaMail session resources objects using a JNDI name.

JAXM Java API for XML Messaging. Enables applications to send and receive document-oriented XML messages using the SOAP standard. These messages can be with or without attachments.

JAXP Java API for XML Processing. A Java API that supports processing of XML documents using DOM, SAX, and XSLT. Enables applications to parse and transform XML documents independent of a particular XML processing implementation.

JAXR Java API for XML Registry. Provides a uniform and standard Java API for accessing different kinds of XML registries. Enables users to build, deploy and discover web services.

JAX-RPC Java API for XML-based Remote Procedure Calls. Enables developers to build interoperable web applications and web services based on XML-based RPC protocols.

JDBC Java Database Connectivity. A standards-based set of classes and interfaces that enable developers to create data-aware components. JDBC implements methods for connecting to and interacting with data sources in a platform- and vendor-independent way.

JDBC connection pool A pool that combines the JDBC data source properties used to specify a connection to a database with the connection pool properties.

JDBC resource A resource used to connect an application running within the application server to a database using an existing JDBC connection pool. Consists of a JNDI name (which is used by the application) and the name of an existing JDBC connection pool.

JDK Java Development Kit. The software that includes the APIs and tools that developers need to build applications for those versions of the Java platform that preceded the Java 2 Platform. *See also* JDK.

JMS Java Message Service. A standard set of interfaces and semantics that define how a JMS client accesses the facilities of a JMS message service. These interfaces provide a standard way for Java programs to create, send, receive, and read messages.

JMS-administered object A pre-configured JMS object—a connection factory or a destination—created by an administrator for use by one or more JMS clients. The use of administered objects allows JMS clients to be provider-independent; that is, it isolates them from the proprietary aspects of a provider. These objects are placed in a JNDI name space by an administrator and are accessed by JMS clients using JNDI lookups.

JMS client An application (or software component) that interacts with other JMS clients using a JMS message service to exchange messages.

JMS connection factory The JMS administered object a JMS client uses to create a connection to a JMS message service.

JMS destination The physical destination in a JMS message service to which produced messages are delivered for routing and subsequent delivery to consumers. This physical destination is identified and encapsulated by an JMS administered object that a JMS client uses to specify the destination for which it is producing messages and/or from which it is consuming messages.

JMS messages Asynchronous requests, reports, or events that are consumed by JMS clients. A message has a header (to which additional fields can be added) and a body. The message header specifies standard fields and optional properties. The message body contains the data that is being transmitted.

JMS provider A product that implements the JMS interfaces for a messaging system and adds the administrative and control functions needed for a complete product.

JMS Service Software that provides delivery services for a JMS messaging system, including connections to JMS clients, message routing and delivery, persistence, security, and logging. The message service maintains physical destinations to which JMS clients send messages, and from which the messages are delivered to consuming clients.

JNDI Java Naming and Directory Interface. This is a standard extension to the Java platform, providing Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise. As part of the Java Enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services.

JNDI name A name used to access a resource that has been registered in the JNDI naming service.

JRE Java Runtime Environment. A subset of the Java Development Kit (JDK) consisting of the Java virtual machine, the Java core classes, and supporting files that provides runtime support for applications written in the Java programming language. *See also* JDK.

JSP JavaServer Page. A text page written using a combination of HTML or XML tags, JSP tags, and Java code. JSPs combine the layout capabilities of a standard browser page with the power of a programming language.

jspc utility The compiler for JSPs. It checks all JSPs for compliance with the JSP specification.

JTA Java Transaction API. An API that allows applications and J2EE servers to access transactions.

JTS Java Transaction Service. The Java service for processing transactions.

key-pair file *See* trust database.

LDAP Lightweight Directory Access Protocol. LDAP is an open directory access protocol that runs over TCP/IP. It is scalable to a global size and millions of entries. Using Sun ONE Directory Server, a provided LDAP server, you can store all of your enterprise's information in a single, centralized repository of directory information that any application server can access through the network.

LDIF LDAP Data Interchange Format. Format used to represent Sun ONE Directory Server entries in text form.

lifecycle event A stage in the server life cycle, such as startup or shutdown.

lifecycle module A module that listens for and performs its tasks in response to events in the server life cycle.

Listener A class, registered with a posting object, that says what to do when an event occurs.

local database connection The transaction context in a local connection is local to the current process and to the current data source, not distributed across processes or across data sources.

local interface An interface that provides a mechanism for a client that is located in the same Java Virtual Machine (JVM) with a session or entity bean to access that bean.

local session A user session that is only visible to one server.

local transaction A transaction that is native to one database and is restricted within a single process. Local transactions work only against a single backend. Local transactions are typically demarcated using JDBC APIs. *See also* global transaction.

mapping The ability to tie an object-oriented model to a relational model of data, usually the schema of a relational database. The process of converting a schema to a different structure. Also refers to the mapping of users to security roles.

MDB *See* message-driven bean.

message-driven bean An enterprise bean that is an asynchronous message consumer. A message-driven bean has no state for a specific client, but its instance variables may contain state across the handling of client messages, including an open database connection and an object reference to an EJB object. A client accesses a message-driven bean by sending messages to the destination for which the message-driven bean is a message listener.

messaging A system of asynchronous requests, reports, or events used by enterprise applications that allows loosely coupled applications to transfer information reliably and securely.

metadata Information about a component, such as its name, and specifications for its behavior.

management information base (MIB) A tree-like structure that defines the variables the master SNMP agent can access. The MIB provides access to the HTTP server's network configuration, status, and statistics. Using SNMP, you can view this information from the network management workstation (NMS). *See also* network management station (NMS) and SNMP.

MIME Data Type MIME (Multi-purpose Internet Mail Extension) types control what types of multimedia files your system supports.

module A web application, enterprise bean, message-driven bean, application client, or connector that has been deployed individually, outside an application. *See also* application, component, lifecycle module.

network management station (NMS) A machine used to remotely manage a specific network. Usually, the NMS software will provide a graph to display collected data or use that data to make sure the server is operating within a particular tolerance. *See also* SNMP.

NTV Name, Type, Value.

object persistence *See* persistence.

O/R mapping tool Object-to-relational [database] tool. A mapping tool within the Sun ONE Application Server Administrative interface that creates XML deployment descriptors for entity beans.

package A collection of related classes that are stored in a common directory. They are often literally packaged together in a Java archive JAR file. *See also* assembly, deployment.

parameter A name/value pair sent from the client, including form field data, HTTP header information, and so on, and encapsulated in a request object. Contrast with attribute. More generally, an argument to a Java method or database-prepared command.

passivation A method of releasing a bean's resources from memory without destroying the bean. In this way, a bean is made to be persistent, and can be recalled without the overhead of instantiation.

permission A set of privileges granted or denied to a user or group. *See also* ACL.

persistence For enterprise beans, the protocol for transferring the state of an entity bean between its instance variables and an underlying database. Opposite of transience. For sessions, the session storage mechanism.

persistence manager The entity responsible for the persistence of the entity beans installed in the container.

persistent state Where the state of an object is kept in persistent storage, usually a database.

pluggable authentication A mechanism that allows J2EE applications to use the Java Authentication and Authorization Service (JAAS) feature from the J2SE platform. Developers can plug in their own authentication mechanisms.

point-to-point delivery model Producers address messages to specific queues; consumers extract messages from queues established to hold their messages. A message is delivered to only one message consumer.

pooling The process of providing a number of preconfigured resources to improve performance. If a resource is pooled, a component can use an existing instance from the pool rather than instantiating a new one. In the Sun ONE Application Server, database connections, servlet instances, and enterprise bean instances can all be pooled.

POP3 Post Office Protocol

prepared command A database command (in SQL) that is precompiled to make repeated execution more efficient. Prepared commands can contain parameters. A prepared statement contains one or more prepared commands.

prepared statement A class that encapsulates a `QUERY`, `UPDATE`, or `INSERT` statement that is used repeatedly to fetch data. A prepared statement contains one or more prepared commands.

presentation layout The format of web page content.

presentation logic Activities that create a page in an application, including processing a request, generating content in response, and formatting the page for the client. Usually handled by a web application.

primary key The unique identifier that enables the client to locate a particular entity bean.

primary key class name A variable that specifies the fully qualified class name of a bean's primary key. Used for JNDI lookups.

principal The identity assigned to an entity as a result of authentication.

private key *See* public key cryptography.

process Execution sequence of an active program. A process is made up of one or more threads.

programmatic security The process of controlling security explicitly in code rather than allowing the component's container (for instance, a bean's container or a servlet engine) to handle it. Opposite of declarative security.

programmer-demarcated transaction *See* bean-managed transaction.

property A single attribute that defines the behavior of an application component. In the `server.xml` file, a property is an element that contains a name/value pair.

public key cryptography A form of cryptography in which each user has a public key and a private key. Messages are sent encrypted with the receiver's public key; the receiver decrypts them using the private key. Using this method, the private key never has to be revealed to anyone other than the user.

publish/subscribe delivery model Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to a topic. The system distributes messages arriving from a topic's multiple publishers to its multiple subscribers.

QOS QOS (Quality of Service) refers to the performance limits you set for a server instance or virtual server. For example, if you are an ISP, you might want to charge different amounts of money for virtual servers depending on how much bandwidth is provided. You can limit two areas: the amount of bandwidth and the number of connections.

queue An object created by an administrator to implement the point-to-point delivery model. A queue is always available to hold messages even when the client that consumes its messages is inactive. A queue is used as an intermediary holding place between producers and consumers.

RAR file Resource ARchive. A JAR archive that contains a resource adapter.

RDB Relational database.

RDBMS Relational database management system.

read-only bean An entity bean that is never modified by an EJB client. *See also* entity bean.

realm A scope over which a common security policy is defined and enforced by the security administrator of the security service. Also called a *security policy domain* or *security domain* in the J2EE specification.

remote interface One of two interfaces for an Enterprise JavaBean. The remote interface defines the business methods callable by a client.

request object An object that contains page and session data produced by a client, passed as an input parameter to a servlet or JavaServer Page (JSP).

resource manager An object that acts as a facilitator between a resource such as a database or message broker, and client(s) of the resource such as Sun ONE Application Server processes. Controls globally-available data sources.

resource reference An element in a deployment descriptor that identifies the component's coded name for the resource.

response object An object that references the calling client and provides methods for generating output for the client.

ResultSet An object that implements the `java.sql.ResultSet` interface. `ResultSets` are used to encapsulate a set of rows retrieved from a database or other source of tabular data.

reusable component A component created so that it can be used in more than one capacity, for instance, by more than one resource or application.

RMI Remote Method Invocation. A Java standard set of APIs that enable developers to write remote interfaces that can pass objects to remote processes.

RMIC Remote Method Invocation Compiler.

role A functional grouping of subjects in an application, represented by one or more groups in a deployed environment. *See also* user, group.

rollback Cancellation of a transaction.

row A single data record that contains values for each column in a table.

RowSet An object that encapsulates a set of rows retrieved from a database or other source of tabular data. `RowSet` extends the `java.sql.ResultSet` interface, enabling `ResultSet` to act as a JavaBeans component.

RPC Remote Procedure Call. A mechanism for accessing a remote object or service.

runtime system The software environment in which programs run. The runtime system includes all the code necessary to load programs written in the Java programming language, dynamically link native methods, manage memory, and handle exceptions. An implementation of the Java virtual machine is included, which may be a Java interpreter.

SAF Server Application Function. A function that participates in request processing and other server activities

schema The structure of the underlying database, including the names of tables, the names and types of columns, index information, and relationship (primary and foreign key) information.

Secure Socket Layer *See* SSL.

security A screening mechanism that ensures that application resources are only accessed by authorized clients.

serializable object An object that can be deconstructed and reconstructed, which enables it to be stored or distributed among multiple servers.

server instance A Sun ONE Application Server can contain multiple instances in the same installation on the same machine. Each instance has its own directory structure, configuration, and deployed applications. Each instance can also contain multiple virtual servers. *See also* virtual server.

servlet An instance of the `Servlet` class. A servlet is a reusable application that runs on a server. In the Sun ONE Application Server, a servlet acts as the central dispatcher for each interaction in an application by performing presentation logic, invoking business logic, and invoking or performing presentation layout.

servlet engine An internal object that handles all servlet metafunctions. Collectively, a set of processes that provide services for a servlet, including instantiation and execution.

servlet runner The part of the servlet engine that invokes a servlet with a request object and a response object. *See* servlet engine.

session An object used by a servlet to track a user's interaction with a web application across multiple HTTP requests.

session bean An enterprise bean that is created by a client; usually exists only for the duration of a single client-server session. A session bean performs operations for the client, such as calculations or accessing other EJBs. While a session bean may be transactional, it is not recoverable if a system crash occurs. Session bean objects can be either stateless (not associated with a particular client) or stateful (associated with a particular client), that is, they can maintain conversational state across methods and transactions. *See also* stateful session bean, stateless session bean.

session cookie A cookie that is returned to the client containing a user session identifier. *See also* sticky cookie.

session timeout A specified duration after which the Sun ONE Application Server can invalidate a user session. *See* session.

single sign-on A situation where a user's authentication state can be shared across multiple J2EE applications in a single virtual server instance.

SMTP Simple Mail Transport Protocol

SNMP SNMP (Simple Network Management Protocol) is a protocol used to exchange data about network activity. With SNMP, data travels between a managed device and a network management station (NMS). A managed device is anything that runs SNMP: hosts, routers, your web server, and other servers on your network. The NMS is a machine used to remotely manage that network.

SOAP The Simple Object Access Protocol (SOAP) uses a combination of XML-based data structuring and Hyper Text Transfer Protocol (HTTP) to define a standardized way of invoking methods in objects distributed in diverse operating environments across the Internet.

SQL Structured Query Language. A language commonly used in relational database applications. SQL2 and SQL3 designate versions of the language.

SSL Secure Sockets Layer. A protocol designed to provide secure communications on the Internet.

state **1.** The circumstances or condition of an entity at any given time. **2.** A distributed data storage mechanism which you can use to store the state of an application using the Sun ONE Application Server feature interface `IState2`. *See also* conversational state, persistent state.

stateful session bean A session bean that represents a session with a particular client and which automatically maintains state across multiple client-invoked methods.

stateless session bean A session bean that represents a stateless service. A stateless session bean is completely transient and encapsulates a temporary piece of business logic needed by a specific client for a limited time span.

sticky cookie A cookie that is returned to the client to force it to always connect to the same server process. *See also* session cookie.

stored procedure A block of statements written in SQL and stored in a database. You can use stored procedures to perform any type of database operation, such as modifying, inserting, or deleting records. The use of stored procedures improves database performance by reducing the amount of information that is sent over a network.

streaming A technique for managing how data is communicated through HTTP. When results are streamed, the first portion of the data is available for use immediately. When results are not streamed, the whole result must be received before any part of it can be used. Streaming provides a way to allow large amounts of data to be returned in a more efficient way, improving the perceived performance of the application.

Sun ONE Directory Server The Sun ONE version of Lightweight Directory Access Protocol (LDAP). Every instance of Sun ONE Application Server uses Sun ONE Directory Server to store shared server information, including information about users and groups. *See also* LDAP.

Sun ONE Message Queue The Sun ONE enterprise messaging system that implements the Java Message Service (JMS) open standard: it is a JMS provider.

system administrator The person who administers Sun ONE Application Server software and deploys Sun ONE Application Server applications.

table A named group of related data in rows and columns in a database.

thread An execution sequence inside a process. A process may allow many simultaneous threads, in which case it is multi-threaded. If a process executes each thread sequentially, it is single-threaded.

TLS Transport Layer Security. A protocol that provides encryption and certification at the transport layer, so that data can flow through a secure channel without requiring significant changes to the client and server applications.

topic An object created by an administrator to implement the publish/subscribe delivery model. A topic may be viewed as node in a content hierarchy that is responsible for gathering and distributing messages addressed to it. By using a topic as an intermediary, message publishers are kept separate from message subscribers.

transaction A set of database commands that succeed or fail as a group. All the commands involved must succeed for the entire transaction to succeed.

Transaction Attribute A transaction attribute controls the scope of a transaction.

transaction context A transaction's scope, either local or global. *See* local transaction, global transaction.

transaction isolation level Determines the extent to which concurrent transactions on a database are visible to one-another.

transaction manager An object that controls a global transaction, normally using the XA protocol. *See* global transaction.

Transaction Recovery Automatic or manual recovery of distributed transactions.

transience A protocol that releases a resource when it is not being used. Opposite of persistence.

trust database I security file that contains the public and private keys; also referred to as the key-pair file.

UDDI Universal Description, Discovery, and Integration. Provides worldwide registry of web services for discovery and integration.

URI Universal Resource Identifier. Describes a specific resource at a domain. Locally described as a subset of a base directory, so that `/ham/burger` is the base directory and a URI specifies `toppings/cheese.html`. A corresponding URL would be `http://domain:port/toppings/cheese.html`.

URL Uniform Resource Locator. An address that uniquely identifies an HTML page or other resource. A web browser uses URLs to specify which pages to display. A URL describes a transport protocol (for example, HTTP, FTP), a domain (for example, `www.my-domain.com`), and optionally a URI.

user A person who uses an application. Programmatically, a user consists of a user name, password, and set of attributes that enables an application to recognize a client. *See also* group, role.

user session A series of user application interactions that are tracked by the server. Sessions maintain user state, persistent objects, and identity authentication.

versioning *See* dynamic reloading.

virtual server 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. Also called a virtual host. A web application can be assigned to a specific virtual server. A server instance can have multiple virtual servers. *See also* server instance.

WAR file Web ARchive. A Java archive that contains a web module. WAR files have the.war extension.

web application 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. Sun ONE Application Server also supports some non-Java web application technologies, such as SHTML and CGI.

web cache An Sun ONE Application Server feature that enables a servlet or JSP to cache its results for a specific duration in order to improve performance. Subsequent calls to that servlet or JSP within the duration are given the cached results so that the servlet or JSP does not have to execute again.

web connector plug-in An extension to a web server that enables it to communicate with the Sun ONE Application Server.

web container *See* container.

web module An individually deployed web application. *See* web application.

web server A host that stores and manages HTML pages and web applications, but not full J2EE applications. The web server responds to user requests from web browsers.

Web Server Plugin The web server plugin is an HTTP reverse proxy plugin that allows you to instruct a Sun One Web Server or Sun ONE Application Server to forward certain HTTP requests to another server.

web service A service offered via the web. A self-contained, self-describing, modular application that can accept a request from a system across the Internet or an intranet, process it, and return a response.

WSDL Web Service Description Language. An XML-based language used to define web services in a standardized way. It essentially describes three fundamental properties of a web service: definition of the web service, how to access that web service, and the location of that web service.

XA protocol A database industry standard protocol for distributed transactions.

XML Extensible Markup Language. A language that uses HTML-style tags to identify the kinds of information used in documents as well as to format documents.

Index