Sun Java System Application Server Enterprise Edition 8.2 Upgrade and Migration Guide

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

Part No: 819–4737

070828@18135

Contents

Preface

This guide explains how to upgrade and migrate applications to Sun Java System Application Server 8.2 Enterprise Edition.

This *Sun Java System Application Server Enterprise Edition 8.2 Upgrade and Migration Guide* describes how to upgrade from earlier versions of Application Server to the current version. This guide also explains how to migrate Java^{TM} 2 Platform, Enterprise Edition (J2EE^{TM} platform) applications from earlier versions of the Sun Java SystemApplication Server and other competitive application servers to Sun Java SystemApplication Server 8.2 Enterprise Edition.

This guide also describes differences between Sun Java System Application Server 8.2 and earlier releases of Application Server.

This preface contains information about and conventions for the entire Sun Java System Application Server documentation set.

Application Server Documentation Set

The Application Server documentation set describes deployment planning and system installation. The Uniform Resource Locator (URL) for stand-alone Application Server documentation is `http://docs.sun.com/app/docs/coll/1310.4`. The URL for Sun Java Enterprise System (Java ES) Application Server documentation is `http://docs.sun.com/app/docs/coll/1310.3`. For an introduction to Application Server, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Application Server Documentation Set

Book Title	Description
Release Notes	Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK^{TM}), and database drivers.
Quick Start Guide	How to get started with the Application Server product.
Installation Guide	Installing the software and its components.

TABLE P–1 Books in the Application Server Documentation Set *(Continued)*

Book Title	Description
Deployment Planning Guide	Evaluating your system needs and enterprise to ensure that you deploy the Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying the server are also discussed.
Developer's Guide	Creating and implementing Java 2 Platform, Enterprise Edition (J2EE platform) applications intended to run on the Application Server that follow the open Java standards model for J2EE components and APIs. Includes information about developer tools, security, debugging, deployment, and creating lifecycle modules.
J2EE 1.4 Tutorial	Using J2EE 1.4 platform technologies and APIs to develop J2EE applications.
Administration Guide	Configuring, managing, and deploying Application Server subsystems and components from the Administration Console.
High Availability Administration Guide	Post-installation configuration and administration instructions for the high-availability database.
Administration Reference	Editing the Application Server configuration file, `domain.xml`.
Upgrade and Migration Guide	Migrating your applications to the new Application Server programming model, specifically from Application Server 6.x and 7. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Performance Tuning Guide	Tuning the Application Server to improve performance.
Troubleshooting Guide	Solving Application Server problems.
Error Message Reference	Solving Application Server error messages.
Reference Manual	Utility commands available with the Application Server; written in man page style. Includes the `asadmin` command line interface.

Related Documentation

Application Server can be purchased by itself or as a component of Java ES, a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If you purchased Application Server as a component of Java ES, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.3`. The URL for all documentation about Java ES and its components is `http://docs.sun.com/prod/entsys.5`.

For other Sun Java System server documentation, go to the following:

- Message Queue documentation
- Directory Server documentation
- Web Server documentation

Additionally, the following resources might be useful:

- The J2EE 1.4 Specifications (`http://java.sun.com/j2ee/1.4/docs/index.html`)
- The J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`)
- The J2EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

Default Paths and File Names

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

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value

install-dir	Represents the base installation directory for Application Server.	Sun Java Enterprise System (Java ES) installations on the Solaris^{TM} platform:
		`/opt/SUNWappserver/appserver`
		Java ES installations on the Linux platform:
		`/opt/sun/appserver/`
		Other Solaris and Linux installations, non-root user:
		user's home directory`/SUNWappserver`
		Other Solaris and Linux installations, root user:
		`/opt/SUNWappserver`
		Windows, all installations:
		SystemDrive`:\Sun\AppServer`

domain-root-dir	Represents the directory containing all domains.	Java ES installations on the Solaris platform:
		`/var/opt/SUNWappserver/domains/`
		Java ES installations on the Linux platform:
		`/var/opt/sun/appserver/domains/`
		All other installations:
		install-dir`/domains/`

domain-dir	Represents the directory for a domain. In configuration files, you might see *domain-dir* represented as follows: `${com.sun.aas.instanceRoot}`	*domain-root-dir*`/`*domain-dir*

TABLE P–2 Default Paths and File Names *(Continued)*

Placeholder	Description	Default Value
instance-dir	Represents the directory for a server instance.	*domain-dir*/*instance-dir*

Typographic Conventions

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

TABLE P–3 Typographic Conventions

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

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

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

Symbol Conventions

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

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.

TABLE P–4 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Searching Sun Product Documentation

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

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

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

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

Third-Party Web Site References

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

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-4737.

CHAPTER 1

Application Server Compatibility Issues

Application Server 8.2 Enterprise Edition is binary compatible with Application Server 8.1, 8.0, and 7.x. J2EE applications that run on versions 8.1, 8.0, and 7.x also work on Application Server8.2 except for the incompatibilities listed in this chapter.

The topics in this chapter discuss the incompatibilities in the following areas:

HTTP File Caching

HTTP file caching, which was present in Application Server 8.1 Enterprise Edition, has been discontinued in Application Server 8.2.

`domain.xml` Elements

If you have not configured message-level security providers for a server instance, Application Server 8.1 applies default configurations from the Domain Administration Server (DAS). Application Server 8.2 does not apply default configurations. You need to manually introduce the message-level security providers — `ClientProvider` and `ServerProvider` — for each server instance that wants to use message-level security. If you have upgraded from an older version to Application Server 8.2, the Upgrade tool does not add these missing elements in the `domain.xml` file.

System Properties

The default security policy of Application Server 8.2 does not allow you to change some system properties. For example, in Application Server 7, the read/write permission of `java.util.PropertyPermission` property is `"*", "read,write";`. In Application Server 8.2 the read/write permission for `java.util.PropertyPermission` is `"*", "read";`.

Implicit URL Rewriting

Application Server 6.x supported implicit URL rewriting, in which the web connector plugin parsed the HTML stream being sent to the browser and appended session IDs to attributes such as `href=` and `frame=`. In Application Server 7,8, and Application Server 8.2, this feature is not available. You need to review your applications and use `encodeURL` and `encodeRedirectURL` on every URL that the applications present to clients (such as mobile phones) that do not support cookies.

Web Server Features

The following web-server-specific features are no longer supported in version Application Server 8.2:

- `cgi-bin`, `shtml`
- Simple Network Management Protocol (SNMP) support
- Netscape API (NSAPI) plugin APIs

- Native-content-handling features
- Web server tools (flexanlg, htpasswd)
- HTTP QoS
- Web server configuration files (*.conf, *.acl, mime.types)
- Web server-specific log rotation facility
- Watch dog process (appserv-wdog)

Realms

The upgrade tool transfers the realms and role mapping configurations, any custom realm classes, and file-based user keyfiles for each domain. The XML tag, security-service, defines the realms and role mapping configuration. This tag is defined in sun-server_1_0.dtd and sun-domain_1_0.dtd. For Application Server 7, the tag data resides in the server.xml and for in Application Server 8.2, in domain.xml.

The upgrade tool locates the class file defined for custom realms and makes it available to the Application Server 8.2 environment. The custom realm class is defined in the class name attribute of tag auth-realm. In the security-service tag, the default-realm attribute points to the realm the server is using. It must point to one of the configured auth-realm names. The default realm is file If the class name for default-realm cannot be found, the upgrade tool will log this as an error.

The package names of the security realm implementations have been renamed from com.iplanet.ias.security.auth.realm in Application Server 7 to com.sun.enterprise.security.auth.realm in Application Server 8.2. Custom realms written using the com.iplanet.* classes must be modified.

The com.sun.enterprise.security.AuthenticationStatus class has been removed.

The com.sun.enterprise.security.auth.login.PasswordLoginModule authenticate method implementation has changed as follows:

```
/**
    * Perform authentication decision.
    * <P> Note: AuthenticationStatus and AuthenticationStatusImpl
    * classes have been removed.
    * Method returns silently on success and returns a LoginException
    * on failure.
    *
    * @return void authenticate returns silently on
    *               successful authentication.
    * @throws LoginException on authentication failure.
    *
    */
abstract protected void authenticate()
    throws LoginException;
```

Default Value for the `delegate` Attribute

In Application Server 7, the default value for the optional attribute delegate was false. In Application Server 8.2, this attribute defaults to true. This change means that by default the Web application classloader first delegates to the parent classloader before attempting to load a class by itself.

The `encodeCookies` Property

URL encoding of cookies is performed, if the `encodeCookies` property of the `sun-web-app` element in the `sun-web.xml` file is set to true. In Application Server 7, the default value of the `encodeCookies` property was true. This property was not present in Application Server 8. In Application Server 8.2, the default value is false.

URL encoding of cookies is unnecessary. Setting this property to true is strongly discouraged. This property is provided only for those rare applications that depended on this behavior in Application Server 7.

CORBA Performance Option

In Application Server 7, users were able to specify the following system property to optionally turn on some Object Request Broker (ORB) performance optimization:

```
-Djavax.rmi.CORBA.UtilClass=com.iplanet.ias.util.orbutil.IasUtilDelegate
```

The ORB performance optimization is turned on, by default, in Application Server 8.2. If you are using the preceding system property reference, you must remove it to avoid interfering with the default optimization.

File Formats

In Application Server 8.2, `domain.xml` is the main server configuration file. In Application Server 7, the main server configuration file was `server.xml`. The DTD file of `domain.xml` is found in `lib/dtds/sun-domain_1_1.dtd`. The upgrade tool included in Application Server 8.2 can be used to move from `server.xml`in Application Server 7 to `domain.xml` in Application Server 8.2.

The `lib/dtds/sun-domain_1_1.dtd` file for Application Server 8.2 is fully backward compatible with the corresponding file for Application Server 8, `sun-domain_1_0.dtd`.

In general, the configuration file formats are *not* backward compatible. The following configuration files are *not* supported:

- `*.conf`
- `*.acl`
- `mime.types`
- `server.xml` (replaced by `domain.xml`)

Cluster Scripts

The `clsetup` and `cladmin` scripts in Application Server 7 are not supported in Application Server 8.2. In Application Server 8.2, the `asadmin configure-ha-cluster` command replaces the `clsetup` script, and `asadmin` commands that operate on clusters replace the commands supported by the `cladmin` script. For more information about the `asadmin` commands, see the *Sun Java System Application Server Enterprise Edition 8.2 Reference Manual*.

Primary Key Attribute Values

In Application Server 7, it was possible to change any field (in the Admin Console) or attribute (in the Command Line Interface (CLI)). In Application Server 8.2, a field or attribute that is the primary key of an item cannot be changed. However, an item can be deleted and then recreated with a new primary key value. In most cases, the primary key is a name, ID, reference, or JNDI name. The following table lists the primary keys that cannot be changed.

Note – In the `domain.xml` file, a field or attribute is called an *attribute*, and an item is called an *element*. For more information about `domain.xml`, see the *Sun Java System Application Server Enterprise Edition 8.2 Administration Reference*.

TABLE 1–1 Primary Key Attributes

Item	Primary Key Field or Attribute
`admin-object-resource`	`jndi-name`
`alert-subscription`	`name`
`appclient-module`	`name`
`application-ref`	`ref`
`audit-module`	`name`
`auth-realm`	`name`
`cluster-ref`	`ref`
`cluster`	`name`

TABLE 1–1 Primary Key Attributes *(Continued)*

Item	Primary Key Field or Attribute
`config`	`name`
`connector-connection-pool`	`name`
`connector-module`	`name`
`connector-resource`	`jndi-name`
`custom-resource`	`jndi-name`
`ejb-module`	`name`
`external-jndi-resource`	`jndi-name`
`http-listener`	`id`
`iiop-listener`	`id`
`j2ee-application`	`name`
`jacc-provider`	`name`
`jdbc-connection-pool`	`name`
`jdbc-resource`	`jndi-name`
`jms-host`	`name`
`jmx-connector`	`name`
`lb-config`	`name`
`lifecycle-module`	`name`
`mail-resource`	`jndi-name`
`message-security-config`	`auth-layer`
`node-agent`	`name`
`profiler`	`name`
`element-property`	`name`
`provider-config`	`provider-id`
`resource-adapter-config`	`resource-adapter-name`
`resource-ref`	`ref`
`security-map`	`name`
`server`	`name`
`server-ref`	`ref`

TABLE 1–1 Primary Key Attributes *(Continued)*

Item	Primary Key Field or Attribute
`system-property`	`name`
`thread-pool`	`thread-pool-id`
`virtual-server`	`id`
`web-module`	`name`
`persistence-manager-factory-resource`	`jndi-name`

Command Line Interface: hadbm

The following table lists options for the command line utility `hadbm` that are no longer supported. For more information about the `hadbm` commands, see the *Sun Java System Application Server Enterprise Edition 8.2 Reference Manual*.

TABLE 1–2 Unsupported `hadbm` Options

Option	Unsupported in Subcommands
`--inetdsetup`	Not supported for the `addnodes` subcommand.
`--inetd`	Not supported for the `create` subcommand.
`--inetdsetupdir`	Not supported for the `create` subcommand.
`--configpath`	Not supported for the `create` subcommand.
`--set managementProtocol`	Not supported for the `create` subcommand.

`--set DataDeviceSize`
`--set TotalDatadeviceSizePerNode`
	Not supported for the `create` or `set` subcommand.

Command Line Interface: start-appserv and stop-appserv

The `start-appserv` and `stop-appserv` commands are deprecated. Use of these commands results in a warning. Use `asadmin start-domain` and `asadmin stop-domain` instead.

In Application Server 8.2, the Log Messages to Standard Error field has been removed from the Admin Console. The `log-to-console` attribute in the `domain.xml` file is deprecated and ignored. The `asadmin set` command has no effect on the `log-to-console` attribute. Use the `---verbose` option of the `asadmin start-domain` command to print messages to the window in which you executed the `asadmin start-domain` command. This option works only if you execute the `asadmin start-domain` command on the machine that has the domain you are starting.

Command Line Interface: asadmin

The following sections describe changes to the command line utility `asadmin`:

For more information about the `asadmin` commands, see the *Sun Java System Application Server Enterprise Edition 8.2 Reference Manual*.

asadmin Subcommands

Subcommands are backward compatible except as noted below.

The `reconfig`subcommand is deprecated and ignored.

The following subcommands are not supported in Application Server 8.2:

- `show-instance-status` (use `list-instances`)
- `restart-instance` (use `stop-instance` followed by `start-instance`)
- `configure-session-persistence` (renamed to `configure-ha-persistence`)
- `create-session-store` (renamed to `create-ha-store`)
- `clear-session-store` (renamed to `clear-ha-store`)

The following subcommands are no longer supported in Application Server 8.2. The software license key and web core were removed, and Application Server 8.2 no longer supports controlled functions from web server features.

- `install-license`
- `display-license`
- `create-http-qos`
- `delete-http-qos`
- `create-mime`
- `delete-mime`
- `list-mime`
- `create-authdb`
- `delete-authdb`
- `list-authdbs`
- `create-acl`
- `delete-acl`
- `list-acls`

Error Codes for Start and Stop Subcommands

For Application Server 7, the error codes for the `start` and `stop` subcommands of the `asadmin` command were based on the desired end state. For example, for `asadmin start-domain`, if the domain was already running, the exit code was 0 (success). If domain startup failed, the exit code was 1 (error).

For Application Server 8.2, the exit codes are based on whether the commands execute as expected. For example, the `asadmin start-domain` command returns exit code 1 if the domain is already running or if domain startup fails. Similarly, `asadmin stop-domain` returns exit code 1 if the domain is already not running or cannot be stopped.

Deprecated and Unsupported Options

Options in the following table are deprecated or no longer supported.

TABLE 1–3 Deprecated and Unsupported `asadmin` Options

Option	Deprecated or Unsupported in Subcommands
`--acceptlang`	Deprecated for the `create-virtual-server` subcommand.
`--acls`	Deprecated for the `create-virtual-server` subcommand.
`--adminpassword`	Deprecated for all relevant subcommands. Use `--passwordfile` instead.
`--blockingenabled`	Deprecated for the `create-http-listener` subcommand.
`--configfile`	Deprecated for the `create-virtual-server` subcommand.
`--defaultobj`	Deprecated for the `create-virtual-server` subcommand.
`--domain`	Deprecated for the `stop-domain` subcommand.
`--family`	Deprecated for the `create-http-listener` subcommand.
`--instance`	Deprecated for all remote subcommands. Use `--target` instead.
`--mime`	Deprecated for the `create-virtual-server` subcommand.
`--optionsfile`	No longer supported for any commands.
`--password`	Deprecated for all remote subcommands. Use `--passwordfile` instead.
`--path`	Deprecated for the `create-domain` subcommand. Use `--domaindir` instead.
`--resourcetype`	Deprecated for all relevant subcommands. Use `--restype` instead.
`--storeurl`	No longer supported for any commands.

TABLE 1–3 Deprecated and Unsupported `asadmin` Options *(Continued)*

Option	Deprecated or Unsupported in Subcommands
`--target`	Deprecated for all `jdbc-connection-pool`, `connector-connection-pool`, `connector-security-map`, and `resource-adapter-config` subcommands.
`--type`	Deprecated for all relevant subcommands.

Dotted Names

The following use of dotted names in `asadmin` `get` and `set` subcommands are not backward compatible:

- The default server name is `server` instead of `server1`.
- server_instance`.resource` becomes `domain.resources.resource`.
- server_instance`.app-module` becomes `domain.applications.app-module`.
- Attributes names format is different. For example,`poolResizeQuantity` is now `pool-resize-quantity`.
- Some aliases supported in Application Server 7 are not supported in Application Server 8.2 .

In Application Server 8.2, the `---passwordfile` option of the `asadmin` command does not read the `password.conf` file, and the upgrade tool does not upgrade this file. For information about creating a password file in Application Server 8.2, see the *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

This table displays a one-to-one mapping of the incompatibilities in dotted names between Application Server 7 and 8.2. The compatible dotted names are not listed in this table.

TABLE 1–4 Incompatible Dotted Names Between Versions

Application Server 7 Dotted Names	Application Server 8.2 Dotted Names
server_instance`.http-listener.` *listener_idserver_instance*`.http-service.` `http-listener.`*listener_id*	*server_instance*`.http-service` `.http-listener.`*listener_id* *config_name*`.http-service` `.http-listener.`*listener_id*
server_instance`.orb`*server_instance*`.iiop-service`	*server_instance*`.iiop-service`*config_name* `.iiop-service`
server_instance`.orblistener`*server_instance* `.iiop-listener`	*server_instance*`.iiop-service` `.iiop-listener.`*listener_id* *config_name*`.iiop-service` `.iiop-listener.`*listener_id*

TABLE 1–4 Incompatible Dotted Names Between Versions *(Continued)*

Application Server 7 Dotted Names	Application Server 8.2 Dotted Names
server_instance.jdbc-resource.*jndi_name*	*server_instance*.resources .jdbc-resource.*jndi_name* domain.resources.jdbc-resource.*jndi_name*
server_instance.jdbc-connection-pool.*pool_id*	*server_instance*.resources.jdbc-connection-pool. *pool_id*domain.resources. jdbc-connection-pool.*pool_id*
server_instance.external-jndi-resource. *jndi_nameserver_instance*. jndi-resource.*jndi_name*	*server_instance*.resources. external-jndi-resource .*jndi_name*domain.resources .external.jndi-resource.*jndi_name*
server_instance.custom-resource.*jndi_name*	*server_instance*.resources. custom-resource.*jndi_name* domain.resources.custom-resource.*jndi_name*
server_instance.web-container.logLevel (see note below)	*server_instance*.log-service.module- log-levels.web-container*config_name* .log-service.module-log-levels.web-container
server_instance.web-container. monitoringEnabled (see note below)	*server_instance*.monitoring-service.module- monitoring-levels.web-container*config_name* .monitoring-service.module -monitoring-levels.web-container
server_instance.j2ee-application. *application_nameserver_instance*.application. *application_name*	*server_instance*.applications.j2ee- application.*application_name* domain.applications.j2ee- application.*application_name*
server_instance.ejb-module.*ejb-module_name*	*server_instance*.applications.ejb-module .*ejb-module_name*domain. applications.ejb-module.*ejb-module_name*
server_instance.web-module.*web-module_name*	*server_instance*.applications.web-module .*web-module_name*domain. applications.web-module.*web-module_name*
server_instance.connector- module.*connector_module_name*	*server_instance*.applications.connector -module.*connector_module_name* domain.applications .connector-module.*connector_module_name*
server_instance.lifecycle-module. *lifecycle_module_name*	*server_instance*.applications.lifecycle -module.*lifecycle_module_name* domain.application.lifecycle- module.*lifecycle_module_name*
server_instance.virtual-server-class	N/A*

TABLE 1–4 Incompatible Dotted Names Between Versions *(Continued)*

Application Server 7 Dotted Names	Application Server 8.2 Dotted Names
server_instance.virtual-server.*virtual-server_id*	*server_instance*.http-service.virtual-server. virtual-server_id*config_name* .http-service.virtual-server.virtual-server_id
server_instance.mime.*mime_id*	N/A*
server_instance.acl.*acl_id*	N/A*
server_instance.virtual-server .*virtual-server_id*.auth-db.*auth-db_id*	N/A*
server_instance.authrealm.*realm_idserver_instance*. security-service.authrealm.*realm_id*	*server_instance*.security-service.auth -realm.*realm_idconfig_name*.security- service-auth-realm.*realm_id*
server_instance.persistence-manager- factory-resource.*jndi_nameserver_instance* .resources.persistence-manager- factory-resource.*jndi_name*	*server_instance*.resources.persistence-manager- factory-resource.*jndi_name*domain.resources. persistence-manager- factory-resource.*jndi_name*
server_instance.http-service.acl.*acl_id*	N/A*
server_instance.mail-resource.*jndi_name*	*server_instance*.resources.mail-resource .*jndi_name*domain.resources.mail -resource.*jndi_name*
server_instance.profiler	*server_instance*.java-config.profiler*config_name* .java-config.profiler

* — These attribute names in Application Server 7 do not correspond directly with Application Server 8.2 dotted names.

Tokens in Attribute Values

The `asadmin get` command shows raw values in Application Server 8.2 instead of resolved values as in Application Server 8. These raw values may be tokens. For example, execute the following command:

```
asadmin get domain.log-root
```

The preceding command displays the following value:

```
${com.sun.aas.instanceRoot}/logs
```

Nulls in Attribute Values

In Application Server 8, attributes with no values contained null. This caused problems in attributes that specified paths. In Application Server 8.2, attributes with no values contain empty strings, as they did in Application Server 7.

◆ ◆ ◆ CHAPTER 2

Upgrading an Application Server Installation

The Upgrade tool, which is bundled with Application Server 8.2, replicates the configuration of a previously installed server in the target installation. The Upgrade tool assists in upgrading the configuration, applications, and certificate data from an earlier version of the Application Server to Application Server 8.2. To view a list of the older Application Server versions from which you can upgrade, refer Table 2–1

This chapter discusses the following topics:

Upgrade Overview

The following table shows supported Sun Java System Application Server upgrades. In this table, PE indicates Platform Edition and EE indicates Enterprise Edition.

TABLE 2–1 Supported Upgrade Paths

Source Installation	8.2 Platform Edition	8.2 Enterprise Edition
7.X PE	Not supported	Supported
7.XSE	-	Supported
7.XEE	-	Supported

TABLE 2–1 Supported Upgrade Paths (Continued)

Source Installation	8.2 Platform Edition	8.2 Enterprise Edition
8.0PE	Supported	Supported
8.1PE	Supported	Supported
8.1EE	-	Supported
8.2PE	-	Supported

Note – Upgrading from 8.1 SE to 8.2 EE only involves installing the HADB packages. You do not need to use the upgrade tool .

The Upgrade tool can be launched by issuing the `asupgrade` command or by choosing the Upgrade option in the Application Server Installer.

Upgrade Tool Interfaces

You can use the tool through the command-line interface (CLI) or the GUI.

To use the Upgrade tool in GUI mode, issue the `asupgrade` command with no options.

To run the Upgrade tool in CLI mode, invoke the `asupgrade` command with the `--c/--console` option. You can run the upgrade CLI in the interactive or non-interactive mode. If you supply all required arguments when invoking `asupgrade` on the console, the upgrade is performed in non-interactive mode and no further input is required. For a complete list of `asupgrade` options, refer Table 2–2. If you invoke the tool only with the `--c/--console` option, the tool enters the interactive CLI mode, where the user is asked for a series of inputs.

Upgrade Terminology

The following are important terms related to the upgrade process:

- Source Server: the installation from which you are upgrading to the new version.
- Target Server: the installation to which you are upgrading.
- Domains Root : the directory where the domains are created. This directory, by default, is the location specified as `AS_DEF_DOMAINS_PATH` in `asenv.conf`
- Domain Directory or *domain-dir*: the directory (within the Domains Root) corresponding to a specific domain. All the configuration and other data pertaining to the domain exists in this directory.
- Install Root: the directory where the Application Server is installed.

- Administration User Name: Name of the user who administers the server. This term refers to the admin user of the Application Server installation from which you want to upgrade.
- Password: Administration user's password to access the Domain Administration Server (DAS)(8-character minimum) of the Application Server installation from which you want to upgrade.
- Master Password: SSL certificate database password used in operations such as Domain Administration Server startup. This term refers to the master password of the Application Server installation from which you want to upgrade.

Upgrade Tool Functionality

The Upgrade Tool migrates the configuration, deployed applications, and certificate databases from an earlier version of the Application Server to the current version. The Upgrade Tool does not upgrade the binaries of the Application Server. The installer is responsible for upgrading the binaries. Database migrations or conversions are also beyond the scope of this upgrade process.

Only those instances that do not use Sun Java System Web Server-specific features are upgraded seamlessly. Configuration files related to HTTP path, CGI bin, SHTML, and NSAPI plug-ins are not be upgraded.

Note – Before starting the upgrade process, make sure that both the source server (the server from which you are upgrading) and the target server (the server to which you are upgrading) are stopped.

Migration of Deployed Applications

Application archives (EAR files) and component archives (JAR, WAR, and RAR files) that are deployed in the Application Server 7.x/ 8.0 environment do not require any modification to run on Application Server 8.2.

Applications and components that are deployed in the source server are deployed on the target server during the upgrade. Applications that do not deploy successfully on the target server must be migrated using the Migration Tool or `asmigrate` command, and deployed again manually. For information on migrating applications using the Migration Tool, refer Chapter 6, "Migrating from Application Server 6.x/7.x."

If a domain contains information about a deployed application and the installed application components do not agree with the configuration information, the configuration is migrated as is without any attempt to reconfigure the incorrect configurations.

Upgrade of Clusters

While upgrading a configuration containing clusters inApplication Server 7.x, specify one or more cluster files or the `clinstance.conf` files. In Application Server 8.x, the clusters are defined in the `domain.xml` file and there is no need to specify separate clusters. Another notable difference is that in Application Server 8.x, all the instances within a cluster reside within the same domain and therefore, in the same `domain.xml` file. In Application Server 7.x, the instance forming a cluster could span more than one domain.

Transfer of Certificates and Realm Files

The Upgrade tool transfers certificates from the source certificate database to the target. The tool supports conversion of JKS certificates to NSS certificates. The tool transfers security policies, password files from standard, file-based realms, and custom realm classes. Refer "Before You Upgrade" on page 31 for specific requirements for providing passwords for the certificate databases.

Upgrade Log

An upgrade log records the upgrade activity. The upgrade log file is named as the `upgrade.log` and is created in the domains root where the upgrade is carried out.

Upgrade Rollback

If an upgrade in progress is cancelled, the configuration before the upgrade was started is restored.

Upgrade Scenarios

The following are the three scenarios in which an upgrade is performed:

- Side-by-side Upgrade: The source server and the target server are installed on the same machine , but under different install locations. You can choose to perform this type of upgrade if you wish to have the configuration corresponding to these installations on the same machine in different locations.
- In-place Upgrade: The target server is installed in the same installation location as the source server. You can choose to perform this type of upgrade if you wish to install the configuration (that is, the domains) in the same location as before. In this scenario, you install the binaries in the same location as the existing binaries using the installer. After installation, if you select the Upgrade checkbox during installation, the installer invokes the upgrade tool , pre-populating it with the source domains directory and target server directory. In this type of upgrade, you provide the source domains root and all the domains under this domains root are upgraded. If you have created multiple domains roots that are customized, you need to specify each of these domains root for upgrading the domains under them.

Before You Upgrade

You need to perform the following steps before upgrade in each of the following scenarios:

- Upgrading from Application Server 7.x: You can perform only a side-by-side upgrade, which involves installing the source application server installation and the target installation on the same machine, under different install locations. Before upgrade, you need to:
  1. Ensure that all the Application Server 7.x domains have the same admin credentials.
  2. Use the same admin credentials as that of Application Server 7.x, if you create a domain during the installation of Application Server 8.2 EE.
- Upgrading from Application Server 8.x EE: You can perform a side-by-side upgrade or an in-place upgrade. Before upgrade, you need to check if any of the keystore or certificate databases have any other password other than `changeit`. You need to change all those passwords to `changeit`. You can use the `keytool` utility for the JKS databases and the `certutil` utility for the NSS databases.

Caution – Failure to change the keystore or certificate database passwords could result in the Upgrade Tool certificate transfer process being aborted abnormally.

After the upgrade, if you wish to restore the values of the keystore or certificate database passwords, you can do so manually by using the `keytool` and `certutil` utilities.

Upgrading from the Command Line

The upgrade utility is run from the command line using the following syntax:

```
asupgrade
[--console ]
[--version ]
[--help ]
[--source applicationserver_7.x/8.x_installation]
[--target applicationserver_8.2_installation]
[--domain domain_name]
[--adminuser admin_user]
[--adminpassword admin_password]
[--masterpassword master_password]
[--targetnsspwdfile targetNSS_password_filepath]
[--nsspwdfile NSS_password_filepath]
[--jkspwdfile JKS_password_filepath]
[--capwdfile CA_password_filepath]
[--clinstancefiles file1 [, file2, file3, ... filen]]
```

The following table describes the command options in greater detail, including the short form, the long form, and a description.

TABLE 2–2 asupgrade Utility Command Options

Short Form	Long Form	Description
-c	--console	Launches the upgrade command line utility.
-V	--version	The version of the Upgrade Tool.
-h or -?	--help	Displays the arguments for launching the upgrade utility.
-t	--target	The domains root directory of the Application Server 8.2 EE installation.
-s	--source	The installation directory when source is Application Server 7.x. Domains root directory when the source is Application Server 8.x and an in-place upgrade is done. Domain directory *domain-dir* when the source is Application Server 8.x and a side-by-side upgrade is done.
-d	--domain	The domain name for the migrated certificates.
-a	--adminuser	The user name of the admin user
-w	--adminpassword	The admin password
-m	--masterpassword	Master password
-n	--nsspwdfile	The path to the NSS password file. This file is a plain text file that contains only the password
-e	--targetnsspwdfile	The path to the target NSS password file. This file is a plain text file that contains only the password
-j	--jkspwdfile	The path to the JKS password file. This file is a plain text file that contains only the password
-p	--capwdfile	The path to the CA certificate password file. This file is a plain text file that contains only the password.
-i	--clinstancefiles	The path to the cluster file, if the source installation is Application Server 7.x. The default filename is `$AS_INSTALL/conf/clinstance.conf`.

The following examples show how to use the `asupgrade` command-line utility to upgrade an existing application server installation to Application Server 8.2.

Example 1: Upgrading an Application Server 7 Installation to Application Server 8.2 EE with Prompts for Certificate Migration.

This example shows how to perform a side-by-side upgrade of a Sun Java SystemApplication Server 7 installation to Sun Java System Application Server 8.2. This command prompts you to migrate certificates. If you reply no, certificates are not migrated.

```
asupgrade --source /home/sunas7 --target /home/sjsas8.2/domains
```

Example 2: Upgrading an Application Server 7.1 EE Installation with Clusters and NSS Certificates to Application Server 8.2 EE

This example shows how to upgrade (side-by-side) a Sun Java System Application Server 7.1 EE installation with a cluster to Sun Java System Application Server 8.2 EE. NSS certificates will be migrated, as will the `clinstance.conf` cluster file.

```
asupgrade --source /home/sjsas7.1 --target /home/sjsas8.2/domains --domain domain1
--nsspwdfile /home/sjsas7.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas7.1/config/clinstance.conf
```

After upgrade, node agents for all remote instances are created on the target DAS. These node agents have to copied to the respective host systems and started. For more information, refer "To Upgrade a Node Agent from Application Server 7.x EE" on page 37

Example 3: Upgrading an Application Server 8.1 EE Installation (in-place) with Clusters and NSS Certificates to Application Server 8.2 EE

This example shows how to perform an in-place upgrade of a Sun Java System Application Server 8.1 EE installation with a cluster to Sun Java System Application Server 8.2 EE. NSS certificates will be migrated, as will the `clinstance.conf` cluster file.

```
asupgrade --source /home/sjsas8.1/domains
--target /home/sjsas8.2/domains
--domain domain1
--nsspwdfile /home/sjsas8.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas8.1/config/clinstance.conf
```

After upgrade, node agents for all remote instances are created on the target DAS. These node agents have to copied to the respective host systems and started. For more information, refer "To Upgrade a Node Agent from Application Server 8.1 EE" on page 37

Example 4: Upgrading an Application Server 8.1 EE Installation (side—by—side) with Clusters and NSS Certificates to Application Server 8.2 EE

This example shows how to perform a side-by-side upgrade of a Sun Java System Application Server 8.1 EE installation with a cluster to Sun Java System Application Server 8.2 EE. NSS certificates will be migrated, as will the `clinstance.conf` cluster file.

```
asupgrade --source /home/sjsas8.1/domains/domain1 --target /home/sjsas8.2/domains
--domain domain1 --nsspwdfile /home/sjsas8.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas8.1/config/clinstance.conf
```

Upgrading Through the Wizard

You can start the Upgrade wizard in GUI mode from the command line or from the desktop.

To start the wizard,

- On UNIX, change to the `<install_dir>/bin` directory and type `asupgrade`.

- On Windows, double click the `asupgrade` icon in the `<install_dir>/bin` directory.

If the Upgrade checkbox was selected during the Application Server installation process, the Upgrade Wizard screen automatically displays after the installation completes.

▼ To Use the Upgrade Wizard

1. **In the Source Installation Directory field, enter the location of the existing installation from which to import the configuration.**

   Provide the appropriate value for the Source Installation Directory field depending on the version of your older installation of Application Server and the type of upgrade you want to perform. Valid values for this are:

   - Enter the install root of the Application Server 7.x, if you are upgrading from Application Server 7.x. Example: `/home/sunappserver7.1` for Solaris/Linux users or `C:\ProgramFiles\SunAppserver7.1` for Windows users. You can only perform a side-by-side upgrade if you are upgrading from Application Server 7.x.
   - Enter the domains root if you are performing an in-place upgrade from Application Server 8.x. Example: `/home/sunappserver7.1/domains` for Solaris/Linux users or `C:\ProgramFiles\SunAppserver7.1\domains` for Windows users.
   - Enter the domain directory if you are performing a side—by-side upgrade from Application Server 8.x. Example: `/home/sunappserver7.1/domains/domain1` for Solaris/Linux users or `C:\ProgramFiles\SunAppserver7.1\domains\domain1` for Windows users.

2. **In the Target Installation Directory field, enter the location of the Application Server installation to which to transfer the configuration.**

   If the upgrade wizard was started from the installation (the Upgrade from Previous Version checkbox was checked during the Application Server installation), the default value for this field is the directory to which the Application Server software was just installed. If the Upgrade tool was not invoked through the installer or if you are upgrading from Application server 7.x, you need to provide the domains root of the Application Server 8.2 EE installation as the input to this field. Example: `/home/sunappserver8.2/domains` for Solaris/Linux users or `C:\ProgramFiles\SunAppserver8.2\domains` for Windows users.

3 **If you are upgrading from Application Server 7.x, you need to enter the admin user name, admin password, and master password that you used for theApplication Server 7.x installation. If you have created a domain when you installed the target server (Application Server 8.2), you need to have used the same admin credentials as that was used for Application Server 7.x. If you have multiple Application Server 7.x domains, all of them need to have the same admin credentials. If you are upgrading Application Server 8.x, you need to enter the following values: admin user as `admin`, admin password as `adminadmin`, and master password as `changeit`. If you enter other values, the Upgrade tool ignores those values and assigns the default values to the admin credentials.**

Note – Refer "Before You Upgrade" on page 31 to know what you need to before you upgrade.

---

4 **If you are upgrading a Application Server 7.x Enterprise Edition installation with clusters and no security certificates, click the Next button and continue with step 9 to enter the cluster files information. If you are upgrading Application Server 8.x Enterprise Edition to Application Server 8.2 Enterprise Edition, click the Next button to proceed with the upgrade. The Upgrade tool automatically detects any clusters in the source installation. If security certificates need to be transformed, continue with step 4.**

5 **If the source installation has security certificates that must be transferred, check the Transfer Security Certificates checkbox, press the Next button.**

The Transfer Security Certificates screen displays.

6 **From the Transfer Security Certificates screen, press the Add Domain button to add domains with certificates to be transferred.**

The Add Domain dialog displays.

7 **From the Add Domain dialog, select the domain name that contains the security certificates to migrate and enter the appropriate passwords.**

8 **Click the OK button when done.**

The Transfer Security Certificates screen will be displayed again.

9 **Repeat steps 6 and 7 until all the domains that have certificates to be transferred have been added. Click Next button when done.**

10 **If you are upgrading a Sun Java System Application Server 7.1 Enterprise Edition installation with clusters to Sun Java System Application Server 8.2 Enterprise Edition, the Transfer Cluster Configurations screen will be displayed. Click the Add Cluster button.**

The Select `clinstance.conf` file dialog box will be displayed. Choose `clinstance.conf` file and click the Open button. The `clinstance.conf` file will be added to the list.

11 **Repeat step 9 until all the cluster configuration files that need to be migrated have been added. Press the Next button.**

Repeat this process until all the cluster configuration files that need to be migrated have been added, and press the Next button.

12 **The Upgrade Results panel is displayed showing the status of the upgrade operation.**

13 **Click the Finish button to close the Upgrade Tool when the upgrade process is complete.**

Upgrading Clusters

The Application Server's Upgrade Tool captures cluster details from the `clinstance.conf` file, the cluster configuration file. If more than one cluster has been defined for the Application Server 7.x, multiple `.conf` files could exist prior to the upgrade. The configuration files could have any name, but all would have the `.conf` file extension. If clusters will be included in an upgrade, consider the following points when you are defining `clinstance.conf` files. Instance names in the `clinstance.conf` file must be unique. For example, in Application Server 7.x, machine A could have `server1` and `server2` participating in a cluster. Machine B could also have a `server1` participating in the same cluster. Typically, the `clinstance.conf` file would include `server1` and `server2` of machine A and `server1` of machine B. Application Server 8.1 requires that instance names in a cluster be unique. Therefore, before you upgrade, in the `clinstance.conf`file, you would need to rename `server1` of machine B to a unique name, such as `server3`or `server1ofmachineB`.

Note – You do not have to rename the `server1` instance itself in machine B. You only need to rename the server in the `clinstance.conf` file.

The expectation is that instances participating in the cluster are homogeneous, in the sense that they would have same type of resources and same applications deployed in them. When the upgrade process runs, the instance that is marked as the master instance is picked up for transferring the configuration. If there is no instance marked as the master instance, one of the instances is randomly picked up and used for transferring the configuration. A cluster is created in the DAS, along with instances defined in the `clinstance.conf` file. All these instances participating in this cluster share the same configuration named `cluster_name-config`, where the *cluster_name* is `cluster_0` for the first cluster, `cluster_1` for the next cluster, and so on. Each instance in the cluster has HTTP and IIOP ports set in their system properties. The HTTP port is the port defined in the `clinstance.conf` file as the instance port. IIOP ports are selected from the `iiop-cluster` configuration in the `server.xml` file.

When you are upgrading Application Server 8.x EE to Application Server 8.2 EE, the upgrade tool automatically detects clusters, if any, on the source installation. There is no need to specify the configuration files, in this case.

▼ To Upgrade a Node Agent from Application Server 7.x EE

Server instances that participate in the cluster and that run on a machine that does not have DAS running on it, are created with a node agent named <*host-name*>-<*domain-name*>, where the *host-name* is the name given in the `clinstance.conf` file for that particular instance and the *domain-name* is the name to which this cluster belongs.

After the upgrade process has been completed on the DAS, install Application Server 8.2 on the other machines where clustered instances need to run.

1 **Copy the node-agent directory from DAS machine to client machine under**`install-dir/nodeagents/`**. For example, if your DAS is installed on** `HostA` **and client machine name is** `HostB`**, the upgrade process would have created a node agent named** `HostB_`<*domain_name*> **as the node agent for** `HostB`**. Therefore, copy**`HostB_`<*domain_name*> **from** `HostA`<*AS82_install_dir*>`/nodeagents/HostB_`<*domain_name*> **directory to** `HostB`<*AS82_install_dir*>`/nodeagents`**. After copying, delete the copied node agent directory under** `HostA`**.**

2 **Start the DAS.**

3 **Start the node agent named** `HostB_`<*domain_name*> **on** `HostB`**. The node agent with rendezvous with the DAS and the remote instances are created in** `HostB`**and the deployed applications are copied over.**

▼ To Upgrade a Node Agent from Application Server 8.1 EE

If you are performing an in-place upgrade , you do not have to upgrade the node agents. The same node-agent directory can be used with the upgraded binaries. If you are performing a side-by-side upgrade , perform the following steps:

1 **Install Application Server 8.2 EE on** `Machine B` **in** `/opt/SUNWappserver8.2` **with the default node agent.**

2 **Perform a side-by-side upgrade from the install location of Application Server 8.1 EE.**

3 **Install Application Server 8.2 EE on** `Machine B` **which has a node agent with remote instances referring to a DAS on** `machine A`**. You must install only the node gent on** `Machine B`**.**

4. **After upgrade, verify the** `nodeagent.properties` **file in** `Machine A` **and** `Machine B` **for the** `agent.adminPort` **property . This file must reflect the same value as that of the** `jmx-connector` **port in the corresponding** `node-agent` **element of the** `domain.xml` **file. If not, edit the** `nodeagent.properties` **file accordingly.**

5. **Start the DAS on** `Machine A`**.**

6. **Start the default node agent on** `Machine A`**. It starts up with the instance,** `instance1`**. The** `cluster1` **cluster is partially started after this step.**

7. **On** `Machine B`**, start up the default node agent of that remote instance.**

8. **You can check the upgraded elements by running the** `asadmin` **commands,**`list-node-agents`**(1),** `list-clusters`**(1),** `list-instances`**(1).**

Correcting Potential PE and EE Upgrade Problems

This section addresses the following issues that could occur during an upgrade to Application Server 8.2:

- "Running the `--domaindir` Option on Older Domains" on page 38
- "To Migrate Additional HTTP Listeners Defined on the Source Server to the Target PE Server" on page 39
- "To Migrate Additional HTTP and IIOP Listeners Defined on the Source Server to the Target EE Server" on page 39
- "Eliminating Port Conflict Problems" on page 40
- "Eliminating Problems Encountered When A Single Domain has Multiple Certificate Database Passwords" on page 40
- "Resolving Load balancer Plug-in Problems During Side-by-Side Upgrade" on page 41

Running the `--domaindir` Option on Older Domains

If you have installed Application Server 8.2 and an older Application Server 8.1 in two separate locations, you might want to use the `--domaindir` option on domains created with the previous version of Application Server without actually upgrading the domains to the latest version. In this scenario, you need to update the `startserv` and `stopserv` scripts to ensure that the domains use the latest binaries of Application Server. Change the scripts to point to the `asenv.conf` file from the latest location.

▼ To Migrate Additional HTTP Listeners Defined on the Source Server to the Target PE Server

If additional HTTP listeners have been defined in the PE source server, those listeners need to be added to the PE target server after the upgrade:

1 **Start the Admin Console.**

2 **Expand Configuration.**

3 **Expand HTTP Service.**

4 **Expand Virtual Servers.**

5 **Select <server>.**

6 **In the right hand pane, add the additional HTTP listener name to the HTTP Listeners field.**

7 **Click Save when done.**

▼ To Migrate Additional HTTP and IIOP Listeners Defined on the Source Server to the Target EE Server

If additional HTTP listeners or IIOP listeners have been defined in the source server, the IIOP ports must be manually updated for the target EE servers before any clustered instances are started. For example, `MyHttpListener` was defined as an additional HTTP listener in `server1`, which is part of the cluster. The other instances in the cluster also have the same HTTP listener, because server instances are symmetrical in a cluster. In the target configuration named <*cluster_name*>`-config`, this listener must be added with its port set to a system property, `{myHttpListener_HTTP_LISTENER_PORT}`. In the target server, each server instance in this cluster that uses this configuration would have system property named `myHttpListener_HTTP_LISTENER_PORT`. The value of this property for all server instances is set to the port value in the source server, `server1`. These system properties for these server instances must be manually updated with nonconflicting port numbers before the server is started.

If additional HTTP listeners have been defined in the source server, those listeners need to be added to the target server after the upgrade:

1 **Start the Admin Console.**

2 **Expand Configuration and select the appropriate** <*server*>`-config` **configuration.**

3 Expand HTTP Service.

4 Expand Virtual Servers.

5 Select *<server>***.**

6 In the right hand pane, add the additional HTTP listener name(s) to the HTTP Listeners field.

7 Click Save when done.

Eliminating Port Conflict Problems

After upgrading the source server to Application Server 8.2 EE, start the domain. Start the node agent, which, by default, starts the server instances. Start the Admin Console and verify that these servers are started. If any of the servers are not running, in the *install_dir*/`nodeagents`/*node-agent-name*/*server_name*/`logs/server.log` file, check for failures that are caused by port conflicts. If there any failures due to port conflicts, use the Admin Console and modify the port numbers so there are no more conflicts. Stop and restart the node agent and servers.

While upgrading from Application Server 7.x SE or EE , a port conflict occurs if one of the instances in 7.x servers is the same as the default ports assigned to the default domain created by Application Serve 8.2 installation. Refer the following list for the values of the ports assigned by default when a domain is created in Application Server 8.2 EE If these conditions exist, start the Admin Console after the upgrade and change the port for the server-config's listener to a nonconflicting port number.

Note – The default ports in Application Server 8.2 EE are:

- 8080 for HTTP Instance (DAS instance)
- 7676 for JMS
- 3700 for IIOP
- 8181 for HTTP_SSL.
- 3820 for IIOP_SSL
- 3920 for IIOP_MUTUALAUTH
- 8686 for JMX_ADMIN

Eliminating Problems Encountered When A Single Domain has Multiple Certificate Database Passwords

If the upgrade includes certificates, provide the passwords for the source PKCS12 file and the target JKS keyfile for each domain that contains certificates to be migrated. Since Application

Server 7uses a different certificate store format (NSS) than that of Application Server 8 PE (JSSE), the migration keys and certificates are converted to the new format. Only one certificate database password per domain is supported. If multiple certificate database passwords are used in a single domain, make all of the passwords the same before starting the upgrade. Reset the passwords after the upgrade has been completed.

Resolving Load balancer Plug-in Problems During Side-by-Side Upgrade

While upgrading from Application Server 7.1 EE to Application Server 8.2 EE, during a side-by-side upgrade, you will not be able to point your new 8.2 load balancer plug-in to the old 7.1 web server installation, if the load balancer plug-in is colocated with other Application server components on a single system. You need to install web server again and point the 8.2 load balancer plug-in installation to the instance belonging to the new installation.

Resolving Problems with Shared Components During Side-by-Side Upgrade

If you have performed a side-by-side upgrade from Application Server 7.x with (MQ and HADB) to Application Server 8.2 EE, do not uninstall the older version — Application Server 7.x. The uninstall process removes shared components, such as JAF, JavaMail, and MQ libraries. The missing shared components will cause the Application Server 8.2 EE installation to malfunction. If you want to uninstall Application Server 7.x, remove `SUNWas*` packages that belong to Application Server 7.x by running the `pkgrm` command, and do not run the uninstall script. If you have already uninstalled Application Server 7.x using the uninstall script, copy the shared components manually by running the `pkgadd` command.

Binary and Remote Upgrades

The tool does not update the runtime binaries of the server. The Upgrade tool upgrades the configuration information and deployed applications of a previously installed server. You need to use the Application Server Installer to install the server binary packages. The first step in the upgrade process is to use the Installer to install the target server binaries.

You cannot perform an upgrade if the source and target server file systems, specifically the domain root file system, are not accessible from the same machine. Currently, most of the upgrade is file based. To perform the upgrade, the user who runs the upgrade needs to have Read permissions for the source and target directories and Write permission for the target directory.

◆ ◆ ◆ CHAPTER 3

Migrating J2EE Applications

You use the Migration Tool (`http://www.java.sun.com/j2ee/tools/migration/`) or the `asmigrate` command to migrate applications from competitive application servers. You also use this tool to migrate the applications that do not deploy successfully after upgrading from an older version of Sun Java SystemApplication Server. This tool works on the input archive or source code to translate the runtime deployment descriptors from the source application server format to generate runtime deployment descriptors that are compliant with the latest version. It also parses the JSP and Java source code files (in case of source code input) and provides runtime support for certain custom JSP tags and proprietary APIs.

This chapter addresses the following topics:

- "Understanding Migration" on page 43
- "Migrating the Application by Using the Migration Tool" on page 48
- "Deploying Migrated Applications" on page 49

Understanding Migration

This section describes the need to migrate J2EE applications and the particular files that must be migrated. Following successful migration, a J2EE application is redeployed to the Application Server.

The following topics are addressed:

- "J2EE Components and Standards" on page 44
- "J2EE Application Components" on page 44
- "Why is Migration Necessary?" on page 45
- "What Needs to be Migrated" on page 46
- "Migration Tool and Other Resources" on page 46

J2EE Components and Standards

Sun Java System Application Server 8.2 (hereafter called Application Server) is a J2EE v1.4-compliant server based on the component standards developed by the Java community. By contrast, the Sun Java SystemApplication Server 7 (Application Server 7) is a J2EE v1.3-compliant server and Sun ONE Application Server 6.x (Application Server 6.x) is a J2EE v1.2-compliant server. Between the three J2EE versions, there are considerable differences with the J2EE application component APIs.

The following table characterizes the differences between the component APIs used with the J2EE v1.4-compliant Sun Java System Application Server 8.2, the J2EE v1.3-compliant Sun ONE Application Server 7, and the J2EE v1.2-compliant Sun ONE Application Server 6.x.

TABLE 3–1 Application Server Version Comparison of APIs for J2EE Components

Component API	Sun ONE Application Server 6.x	Sun Java System Application Server 7	Sun Java System Application Server 8.2
JDK	1.2.2	1.4	1.4
Servlet	2.2	2.3	2.4
JSP	1.1	1.2	2.0
JDBC	2.0	2.0	2.1, 3.0
EJB	1.1	2.0	2.0
JNDI	1.2	1.2	1.2.1
JMS	1.0	1.1	1.1
JTA	1.0	1.01	1.01

J2EE Application Components

J2EE simplifies development of enterprise applications by basing them on standardized, modular components, providing a complete set of services to those components, and handling many details of application behavior automatically, without complex programming. J2EE v1.4 architecture includes several component APIs. Prominent J2EE components include:

- Client Application
- Web Application
- Enterprise Java Beans (EJB)
- Connector
- Enterprise Application Archive (EAR)

J2EE components are packaged separately and bundled into a J2EE application for deployment. Each component, its related files such as GIF and HTML files or server-side utility classes, and a deployment descriptor are assembled into a module and added to the J2EE application. A J2EE application is composed of one or more enterprise bean(s), Web, or application client component modules. The final enterprise solution can use one J2EE application or be made up of two or more J2EE applications, depending on design requirements.

A J2EE application and each of its modules has its own deployment descriptor. A deployment descriptor is an XML document with a .xml extension that describes a component's deployment settings.

A J2EE application with all of its modules is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with a .ear extension. The EAR file contains EJB JAR files, application client JAR files and/or Web Archive (WAR) files.

For more information on J2EE, see:

- J2EE 1.4 tutorial
- J2EE overview
- J2EE website

Why is Migration Necessary?

Although J2EE specifications broadly cover requirements for applications, they are nonetheless evolving standards. They either do not cover some aspects of applications or leave implementation details to the application providers.

This leads to different implementations of the application servers, also well as difference in the deployment of J2EE components on application servers. The array of available configuration and deployment tools for use with any particular application server product also contributes to the product implementation differences.

The evolutionary nature of the specifications itself presents challenges to application providers. Each of the component APIs are also evolving. This leads to a varying degree of conformance by products. In particular, an emerging product, such as the Application Server, has to contend with differences in J2EE application components, modules, and files deployed on other established application server platforms. Such differences require mappings between earlier implementation details of the J2EE standard, such as file naming conventions, and messaging syntax.

Moreover, product providers usually bundle additional features and services with their products. These features are available as custom JSP tags or proprietary Java API libraries. Unfortunately, using these proprietary features renders these applications non-portable.

What Needs to be Migrated

The J2EE application consists of the following file categories that need to be migrated:

- Deployment descriptors (XML files)
- JSP source files that contain Proprietary APIs
- Java source files that contain Proprietary APIs

Deployment descriptors (XML files)

Deployment is accomplished by specifying deployment descriptors for standalone enterprise beans (EJB, JAR files), front-end Web components (WAR files) and enterprise applications (EAR files). Deployment descriptors are used to resolve all external dependencies of the J2EE components or applications. The J2EE specification for deployment descriptors is common across all application server products. However, the specification leaves several deployment aspects of components pertaining to an application dependent on product implementation.

JSP source files

J2EE specifies how to extend JSP by adding extra custom tags. Product vendors include some custom JSP extensions in their products, simplifying some tasks for developers. However, usage of these proprietary custom tags results in non-portability of JSP files. Additionally, JSP can invoke methods defined in other Java source files as well. The JSPs containing proprietary APIs need to be rewritten before they can be migrated.

Java source files

The Java source files can be EJBs, servlets, or other helper classes. The EJBs and servlets can invoke standard J2EE services directly. They can also invoke methods defined in helper classes. Java source files are used to encode the business layer of applications, such as EJBs. Vendors bundle several services and proprietary Java API with their products. The use of proprietary Java APIs is a major source of non-portability in applications. Since J2EE is an evolving standard, different products can support different versions of J2EE component APIs.

Migration Tool and Other Resources

The Migration Tool for Sun Java System Application Server 8.2 (hereafter called Migration Tool) migrates J2EE applications from other server platforms to Sun Java System Application Server 8.2.

The following source platforms are supported for Sun Java System Application Server 8.2:

- Sun ONE Application Server 6.*x*
- Sun Java System Application Server 7
- Sun Java System Application Server 8.0/8.1

- J2EE Reference Implementation Application Server (RI) 1.3, 1.4 Beta1
- WebLogic Application Server (WLS) 5.1, 6.0, 6.1, 8.1
- WebSphere Application Server (WAS) 4.0, 5.x
- Sun ONE Web Server 6.0
- JBoss Application Server 3.0
- TomCat Web Server 4.1

Migration Tool automates the migration of J2EE applications to Sun Java System Application Server 8.2, without much modification to the source code.

The key features of the tool are:

- Migration of application server-specific deployment descriptors
- Runtime support for selected custom Java Server Pages (JSP) tags and proprietary APIs
- Conversion of selected configuration parameters with equivalent functionality in Application Server
- Automatic generation of Ant based scripts for building and deploying the migrated application to the target server, Application Server
- Generation of comprehensive migration reports after achieving migration

Download the Migration Tool from the following location:

http://java.sun.com/j2ee/tools/migration/index.html (`http://java.sun.com/j2ee/tools/migration/index.html`).

The Java Application Verification Kit (AVK) for the Enterprise helps build and test applications to ensure that they are using the J2EE APIs correctly and to migrate to other J2EE compatible application servers using specific guidelines and rules.

Download the Java Application Verification Kit (AVK) from the following location:

http://java.sun.com/j2ee/verified/ (`http://java.sun.com/j2ee/verified/`).

Before Migrating the Application

The Migration tool provides limited support for proprietary APIs and custom JSP tags. The rest of the unsupported API usages show up in the Migration report and need to be manually corrected before you can deploy your application. Therefore, before you start the migration, it is recommended that you run the Application Verification Kit (`http://java.sun.com/j2ee/avk/`)(AVK) to gauge the extent of J2EE compliance of your application.

Before you actually begin the migration process, refer `http://java.sun.com/j2ee/tools/migration/doc/prerun.html` for information on how to prepare your application for migration.

Migrating the Application by Using the Migration Tool

You can run the Migration Tool in GUI or command-line mode.

To start the Migration Tool on Solaris and Linux in GUI mode, type the following command at the shell prompt:

```
<install-dir>/bin/asmigrate.sh -u
```

To start the Migration Tool on Solaris and Linux in command—line mode, type the following command at the shell prompt:

```
<install-dir>/bin/asmigrate.sh -c -S <sourceserver> -t <targetdirectory> -T <targetserver> <operand>
or
install-dir>/bin/asmigrate.sh -c -S <sourceserver> -s <sourcedirectory> -t <targetdirectory> -T <targetserver>
```

To start the Migration Tool on Windows, perform the following steps:

1. Open a DOS command-prompt window.
2. Change to the directory in which you have installed the Application Server 8.2.
3. To start the Migration Tool in GUI mode, type the following command at the DOS prompt:

   ```
   asmigrate.cmd -u
   ```

   Or, to start the Migration Tool in command—line mode, type the following command at the DOS prompt:

```
asmigrate.cmd -c -S <sourceserver> -t <targetdirectory> -T <targetserver> <operand>
or
asmigrate.cmd -c -S <sourceserver> -s <sourcedirectory> -t <targetdirectory> -T <targetserver>
```

For more information on the command-line options of the `asmigrate` command or the Migration Tool, refer http://java.sun.com/j2ee/tools/migration/doc/run.html#CLI

For step-by-step instructions on how to use this tool to migrate your application, refer http://java.sun.com/j2ee/tools/migration/doc/StepByStep.html or the Migration Tool Online Help.

After the Migration Tool migrates your J2EE application, you need to analyze the Migration report to know what additional changes you need to make to the generated output application. You make these changes and then deploy the migrated application. For details on how to perform these tasks, refer http://java.sun.com/j2ee/tools/migration/doc/postrun.html.

Deploying Migrated Applications

To be able to deploy your migrated applications on Application Server 8.2, it is important to understand classloaders in Application Server 8.2 and changes to the architecture of Application Server 8.2.

In Application Server 7, the DAS controls multiple local instances. The Common Classloader loads the classes in the *install-dir*/<*yourdomain*>/<*yourinstance*>/`lib/classes` directory and the *install-dir*/<*yourdomain*>/<*yourinstance*>/`lib` directory. All resources and configurations correspond to a specific instance.

In Application Server 8.2, the DAS controls local and remote instances. The Common Classloader loads the JAR and ZIP files in the *domain-dir*/`lib` directory and the classes in the *domain-dir*/`lib/classes` directory.

In Application Server 8.2, any JAR file placed in the `lib` directory of the DAS is replicated to all instances controlled by that DAS. The JAR files bundled with the Application Server reside in the *install-dir*/`lib` directory.

For more information on the classloader hierarchy in Application Server 8.2, see "The Classloader Hierarchy" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

You can use the delegation inversion mechanism to use libraries bundled with your application instead of those bundled with the Application Server. However, it is safe to use this mechanism only for web modules that do not access EJB components and do not interact with other applications. For more information on the delegation model of Application Server 8.2, see "Classloader Delegation" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

Note – The default value of the `delegate` attribute is `true` in Application Server 8.2. See "Default Value for the `delegate` Attribute" on page 16.

The JAXP 1.3 parser is bundled with Application Server 8.2. You cannot override the JAXP 1.3 parser for Application Server 8.2.

In Application Server 8.2, to share a library with all the applications and modules in a domain, place the libraries (JAR files) in the *domain-dir*/`lib` directory and restart the Application Server. The Common Classloader will load the new libraries. Use this approach to share commonly shared libraries, such as JDBC drivers.

To share libraries across a specific cluster instead of over an entire domain, add the JAR files to the *domain-dir*/`config`/<*cluster-name*>`-config/lib` directory and add the path to the JAR files in the `classpath-suffix` attribute. For instructions on how to change this attribute, see "Using the System Classloader" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

Copy the JAR files to *domain-dir*/`config`/<*cluster-name*>`-config`/`lib`/`ext` directory to add to `java.ext.dirs`. To create an optional package that can be shared across the domain, add the JAR file to *domain-dir*/`lib`/`ext` directory and restart the Application Server.

Note – If multiple applications deployed on a single instance require different versions of the same JAR file, ensure that those JAR files have different names.

◆ ◆ ◆ CHAPTER 4

Migrating from EJB 1.1 to EJB 2.0

Although the EJB 1.1 specification will continue to be supported in Sun Java System Application Server 8.2, the use of the EJB 2.0 architecture is recommended, so that you can leverage its enhanced capabilities.

To migrate EJB 1.1 to EJB 2.0 you need to make several modifications, including a few within the source code of the components.

Essentially, the required modifications relate to the differences between EJB 1.1 and EJB 2.0, all of which are described in the following topics.

- "EJB Query Language" on page 51
- "Local Interfaces" on page 52
- "EJB 2.0 Container-Managed Persistence (CMP)" on page 52
- "Migrating EJB Client Applications" on page 54
- "Migrating CMP Entity EJBs" on page 56

EJB Query Language

The EJB 1.1 specification left the manner and language for forming and expressing queries for finder methods to each individual application server. While many application server vendors let developers form queries using SQL, others use their own proprietary language specific to their particular application server product. This mixture of query implementations causes inconsistencies between application servers.

The EJB 2.0 specification introduces a query language called *EJB Query Language*, or *EJB QL* to correct many of these inconsistencies and shortcomings. EJB QL is based on SQL92. It defines query methods, in the form of both finder and select methods, specifically for entity beans with container-managed persistence. EJB QL's principal advantage over SQL is its portability across EJB containers and its ability to navigate entity bean relationships.

Local Interfaces

In the EJB 1.1 architecture, session and entity beans have one type of interface, a remote interface, through which they can be accessed by clients and other application components. The remote interface is designed such that a bean instance has remote capabilities; the bean inherits from RMI and can interact with distributed clients across the network.

With EJB 2.0, session beans and entity beans can expose their methods to clients through two types of interfaces: a *remote interface* and a *local interface*. The 2.0 remote interface is identical to the remote interface used in the 1.1 architecture, whereby, the bean inherits from RMI, exposes its methods across the network tier, and has the same capability to interact with distributed clients.

However, the local interfaces for session and entity beans provide support for lightweight access from EJBs that are local clients; that is, clients co-located in the same EJB container. The EJB 2.0 specification further requires that EJBs that use local interfaces be within the same application. That is, the deployment descriptors for an application's EJBs using local interfaces must be contained within one `ejb-jar` file.

The local interface is a standard Java interface. It does not inherit from RMI. An enterprise bean uses the local interface to expose its methods to other beans that reside within the same container. By using a local interface, a bean may be more tightly coupled with its clients and may be directly accessed without the overhead of a remote method call.

In addition, local interfaces permit values to be passed between beans with pass by reference semantics. Because you are now passing a reference to an object, rather than the object itself, this reduces the overhead incurred when passing objects with large amounts of data, resulting in a performance gain.

EJB 2.0 Container-Managed Persistence (CMP)

The EJB 2.0 specification expanded CMP to allow multiple entity beans to have relationships among themselves. This is referred to as *Container-Managed Relationships* (CMR). The container manages the relationships and the referential integrity of the relationships.

The EJB 1.1 specification presented a more limited CMP model. The EJB 1.1 architecture limited CMP to data access that is independent of the database or resource manager type. It allowed you to expose only an entity bean's instance state through its remote interface; there is no means to expose bean relationships. The EJB 1.1 version of CMP depends on mapping the instance variables of an entity bean class to the data items representing their state in the database or resource manager. The CMP instance fields are specified in the deployment descriptor, and when the bean is deployed, the deployer uses tools to generate code that implements the mapping of the instance fields to the data items.

You must also change the way you code the bean's implementation class. According to the EJB 2.0 specification, the implementation class for an entity bean that uses CMP is now defined as an abstract class.

The following topics are discussed in this section:

- "Defining Persistent Fields" on page 53
- "Defining Entity Bean Relationships" on page 53
- "Message-Driven Beans" on page 53

Defining Persistent Fields

The EJB 2.0 specification lets you designate an entity bean's instance variables as CMP fields or CMR fields. You define these fields in the deployment descriptor. CMP fields are marked with the element `cmp-field`, while container-managed relationship fields are marked with the element `cmr-field`.

In the implementation class, note that you do not declare the CMP and CMR fields as public variables. Instead, you define `get` and `set` methods in the entity bean to retrieve and set the values of these CMP and CMR fields. In this sense, beans using the 2.0 CMP follow the JavaBeans model: instead of accessing instance variables directly, clients use the entity bean's `get` and `set` methods to retrieve and set these instance variables. Keep in mind that the `get` and `set` methods only pertain to variables that have been designated as CMP or CMR fields.

Defining Entity Bean Relationships

As noted previously, the EJB 1.1 architecture does not support CMRs between entity beans. The EJB 2.0 architecture does support both one-to-one and one-to-many CMRs. Relationships are expressed using CMR fields, and these fields are marked as such in the deployment descriptor. You set up the CMR fields in the deployment descriptor using the appropriate deployment tool for your application server.

Similar to CMP fields, the bean does not declare the CMR fields as instance variables. Instead, the bean provides `get` and `set` methods for these fields.

Message-Driven Beans

Message-driven beans are another new feature introduced by the EJB 2.0 architecture. Message-driven beans are transaction-aware components that process asynchronous messages delivered through the Java Message Service (JMS). The JMS API is an integral part of the J2EE 1.3 and J2EE 1.4 platform.

Asynchronous messaging allows applications to communicate by exchanging messages so that senders are independent of receivers. The sender sends its message and does not have to wait for the receiver to receive or process that message. This differs from synchronous communication, which requires the component that is invoking a method on another component to wait or block until the processing completes and control returns to the caller component.

Migrating EJB Client Applications

This section includes the following topics:

- "Declaring EJBs in the JNDI Context" on page 54
- "Recap on Using EJB JNDI References" on page 55

Declaring EJBs in the JNDI Context

In Sun Java System Application Server 8.2, EJBs are systematically mapped to the JNDI sub-context *ejb/*. If you attribute the JNDI name *Account* to an EJB, the Sun Java System Application Server 8.2 automatically creates the reference *ejb/Account* in the global JNDI context. The clients of this EJB therefore have to look up *ejb/Account* to retrieve the corresponding home interface.

Let us examine the code for a servlet method deployed in Sun ONE Application Server 6.x.

The servlet presented here calls on a stateful session bean, BankTeller, mapped to the root of the JNDI context. The method whose code you are considering is responsible for retrieving the home interface of the EJB, to enable a BankTeller object to be instantiated, and a remote interface for this object to be retrieved, so that you can make business method calls to this component.

```
/**
   * Look up the BankTellerHome interface using JNDI.
   */
private BankTellerHome lookupBankTellerHome(Context ctx)
      throws NamingException
{
    try
    {
      Object home = (BankTellerHome) ctx.lookup("ejb/BankTeller");
      return (BankTellerHome) PortableRemoteObject.narrow(home,
                  BankTellerHome.class);
    }
    catch (NamingException ne)
    {
      log("lookupBankTellerHome: unable to lookup BankTellerHome" +
            "with JNDI name ’BankTeller’: " + ne.getMessage() );
        throw ne;
    }
}
```

As the code already uses `ejb/BankTeller` as an argument for the lookup, there is no need for modifying the code to be deployed on Sun Java System Application Server 8.2.

Recap on Using EJB JNDI References

This section summarizes the considerations when using EJB JNDI references. Where noted, the consideration details are specific to a particular source application server platform.

Placing EJB References in the JNDI Context

It is only necessary to modify the name of the EJB references in the JNDI context mentioned above (moving these references from the JNDI context root to the sub-context *ejb/*) when the EJBs are mapped to the root of the JNDI context in the existing WebLogic application.

If these EJBs are already mapped to the JNDI sub-context `ejb/` in the existing application, no modification is required.

However, when configuring the JNDI names of EJBs in the deployment descriptor within the Sun Java Studio IDE, it is important to avoid including the prefix `ejb/` in the JNDI name of an EJB. Remember that these EJB references are *automatically* placed in the JNDI `ejb/` sub-context with Sun Java System Application Server 8.2. So, if an EJB is given to the JNDI name *BankTeller* in its deployment descriptor, the reference to this EJB will be translated by Sun Java System Application Server 8.2 into `ejb/BankTeller`, and this is the JNDI name that client components of this EJB must use when carrying out a lookup.

Global JNDI context versus local JNDI context

Using the global JNDI context to obtain EJB references is a perfectly valid, feasible approach with Sun Java System Application Server 8.2. Nonetheless, it is preferable to stay as close as possible to the J2EE specification, and retrieve EJB references through the local JNDI context of EJB client applications. When using the local JNDI context, you must first declare EJB resource references in the deployment descriptor of the client part (`web.xml` for a Web application, `ejb-jar.xml` for an EJB component).

Migrating CMP Entity EJBs

This section describes the steps to migrate your application components from the EJB 1.1 architecture to the EJB 2.0 architecture.

In order to migrate a CMP 1.1 bean to CMP 2.0, we first need to verify if a particular bean can be migrated. The steps to perform this verification are as follows.

▼ To Verify if a Bean Can be Migrated

1 **From the `ejb-jar.xml` file, go to the `<cmp-fields>` names and check if the optional tag `<prim-key-field>` is present in the `ejb-jar.xml file` and has an indicated value. If it does, go to next step.**

Look for the `<prim-key-class>` field name in the `ejb-jar.xml`, get the class name, and get the `public instance variables` declared in the class. Now see if the signature (name and case) of these variables matches with the `<cmp-field>` names above. Segregate the ones that are found. In these segregated fields, check if some of them start with an upper case letter. If any of them do, then migration cannot be performed.

2 **Look into the bean class source code and obtain the java types of all the `<cmp-field>` variables.**

3 **Change all the `<cmp-field>` names to lowercase and construct accessors from them. For example if the original field name is `Name` and its java type is `String`, the accessor method signature is:**

```
Public void setName(String name)Public String getName()
```

4 **Compare these accessor method signatures with the method signatures in the bean class. If an exact match is found, migration is not possible.**

5 **Get the custom finder methods signatures and their corresponding SQLs. Check if there is a Join, Outer join, or an OrderBy in the SQL. If yes, you cannot migrate, because EJB QL does not support Join, Outer join, orOrderBy.**

6 **Any CMP 1.1 finder, which used `java.util.Enumeration`, must now use `java.util.Collection`. Change your code to reflect this. CMP2.0 finders cannot return `java.util.Enumeration`.**

"Migrating the Bean Class" on page 56 explains how to perform the actual migration process.

Migrating the Bean Class

This section describes the steps required to migrate the bean class to Sun Java System Application Server 8.2.

▼ To Migrate the Bean Class

1. **Prepend the bean class declaration with the keyword `abstract`.**

   For example if the bean class declaration was:

   ```
   public class CabinBean implements EntityBean
   ```

   change it to:

   ```
   abstract public class CabinBean implements EntityBean
   ```

2. **Prefix the accessors with the keyword `abstract`.**

3. **Insert all the accessors after modification into the source (.java) file of the bean class at class level.**

4. **Comment out all the `cmp` fields in the source file of the bean class.**

5. **Construct protected instance variable declarations from the `cmp-field` names in lowercase and insert them at the class level.**

6. **Read up all the `ejbCreate()` method bodies (there could be more than one `ejbCreate`).**

   Look for the pattern ”`<cmp-field>=`*some value or local variable*’, and replace it with the expression ”abstract mutator method name (*same value or local variable*)’.

   For example, if the `ejbCreate` body before migration is:

   ```
   public MyPK ejbCreate(int id, String name) {
      this.id = 10*id;
      Name = name;   //1
      return null;
   }
   ```

   Change it to:

   ```
   public MyPK ejbCreate(int id, String name) {
      setId(10*id);
      setName(name);   //1
      return null;
   }
   ```

   Note that the method signature of the abstract accessor in `//1` is as per the Camel Case convention mandated by the EJB 2.0 specification. Also, the keyword ”*this*’ may or may not be present in the original source, but it *must be removed* from the modified source file.

7 **Initialize all the protected variables declared in the `ejbPostCreate()` methods in step 5.**

The protected variables will be equal in number with the `ejbCreate()` methods. This initialization will be done by inserting the initialization code in the following manner:

```
protected String name;  //from step 5
protected int id;  //from step 5
public void ejbPostCreate(int id, String name) {
   name = getName();    /*abstract accessor*/ //inserted in this step
   id  = getId();        /*abstract accessor*/ //inserted in this step
}
```

8 **Inside the `ejbLoad` method, set the protected variables to the beans' database state.**

To do so, insert the following lines of code:

```
public void ejbLoad() {
   name = getName();    // inserted in this step
   id = getId();        // inserted in this step
   ...                  // existing code
}
```

9 **Similarly, update the bean's state inside `ejbStore()` so that its database state gets updated.**

But remember, you are not allowed to update the setters that correspond to the primary key outside the `ejbCreate()`, so do not include them inside this method. Insert the following lines of code:

```
public void ejbStore() {
    setName(name);       //inserted in this step
    setId(id);           //Do not insert this if
                         //it is a part of the
                         //primary key.
    ...                  //already present code
}
```

10 **Replace all occurrences of any `<cmp-field>` variable names with the equivalent protected variable name (as declared in step 5).**

If you do not migrate the bean, at the minimum you need to insert the `<cmp-version>1.`*x*`</cmp-version>` tag inside the `ejb-jar.xml` file at the appropriate place, so that the unmigrated bean still works on Sun Java System Application Server 8.2.

Migration of ejb-jar.xml

To migrate the file `ejb-jar.xml` to Sun Java System Application Server 8.2, perform the following steps:

▼ To Migrate the EJB Deployment Descriptor

To migrate the EJB deployment descriptor file, `ejb-jar.xml`, edit the file and make the following changes.

1. **Convert all `<cmp-fields>` to lowercase.**

2. **Insert the tag `<abstract-schema-name>` after the `<reentrant>` tag.**

   The schema name will be the name of the bean as in the `< ejb-name>` tag, prefixed with `ias_`.

3. **Insert the following tags after the `<primkey-field>` tag:**

   ```
   <security-identity>
      <use-caller-identity/>
   </security-identity>
   ```

4. **Use the SQL obtained above to construct the EJB QL from SQL.**

5. **Insert the `<query>` tag and all its nested child tags with all the required information just after the `<security-identity>` tag.**

Custom Finder Methods

The custom finder methods are the `findBy` methods (other than the default `findByPrimaryKey` method), which can be defined in the home interface of an entity bean. Since the EJB 1.1 specification does not stipulate a standard for defining the logic of these finder methods, EJB server vendors are free to choose their implementations. As a result, the procedures used to define the methods vary considerably between the different implementations chosen by vendors.

Sun ONE Application Server 6.x uses standard SQL to specify the finder logic.

Information concerning the definition of this finder method is stored in the enterprise bean's persistence descriptor (`Account-ias-cmp.xml`) as follows:

```
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerSQL</name>
    <type>java.lang.String</type>
    <value>
       SELECT BRANCH_CODE,ACC_NO FROM ACCOUNT where CUST_NO = ?
    </value>
    <delimiter>,</delimiter>
  </property>
</bean-property>
<bean-property>
```

```
  <property>
    <name>findOrderedAccountsForCustomerParms</name>
    <type>java.lang.Vector</type>
    <value>CustNo</value>
    <delimiter>,</delimiter>
  </property>
</bean-property>
```

Each `findXXX` finder method therefore has two corresponding entries in the deployment descriptor (SQL code for the query, and the associated parameters).

In Sun Java System Application Server 8.2 the custom finder method logic is also declarative, but is based on the EJB query language EJB QL.

The EJB-QL language cannot be used on its own. It has to be specified inside the file `ejb-jar.xml`, in the `<ejb-ql>` tag. This tag is inside the `<query>` tag, which defines a query (finder or select method) inside an EJB. The EJB container can transform each query into the implementation of the finder or select method. Here is an example of an `<ejb-ql>` tag:

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>hotelEJB</ejb-name>
      ...
      <abstract-schema-name>TMBankSchemaName</abstract-schema-name>
      <cmp-field>
      ...
      <query>
        <query-method>
          <method-name>findByCity</method-name>
            <method-params>
              <method-param>java.lang.String</method-param>
            </method-params>
        </query-method>
        <ejb-ql>
          <![CDATA[SELECT OBJECT(t) FROM TMBankSchemaName AS t
                                        WHERE t.city = ?1]]>
        </ejb-ql>
      </query>
    </entity>
    ...
  </enterprise-beans> ...
</ejb-jar>
```

◆ ◆ ◆ CHAPTER 5

J2EE 1.4 Compatibility Issues

The following topics are covered in this chapter:

Binary Compatibility

The Java SDK included in Application Server 8.2 is the J2EE version 1.4 SDK. This version of the J2EE SDK is compatible with J2EE SDK, v1.3.

Source Compatibility

Downward source compatibility is not supported. If source files use new J2EE APIs, they are not usable with an earlier version of the J2EE platform.

In general, the policy is as follows:

- Maintenance releases do not introduce any new APIs, so they maintain source-compatibility with one another. However, since J2EE is based on J2SE, a new Application Server release may include a new version of J2SE. For more information, refer to the J2SE document on compatibility issues:

  http://java.sun.com/j2se/1.4.2/compatibility.html (`http://java.sun.com/j2se/1.4.2/compatibility.html`)
- Functionality releases and major releases maintain upwards but not downwards source-compatibility.

Deprecated APIs are methods and classes that are supported only for backward compatibility, and the compiler generates a warning message whenever one of these is used, unless the `-nowarn` command-line option is used. It is recommended that programs be modified to eliminate the use of deprecated methods and classes, though there are no plans to remove such methods and classes entirely.

Incompatibilities with the J2EE 1.4 Platform (since the J2EE 1.3 release)

The Sun Java System Application Server 8.2 release is based on the Java 2 Platform, Enterprise Edition, version 1.4. The Sun Java System Application Server 7 release is based on the Java 2 Platform, Enterprise Edition, version 1.3.

Almost all existing programs must run on the Sun Java System Application Server 8.2 release without modification. However, there are some minor potential incompatibilities that involve rare circumstances and corner cases that have been documented here for completeness.

- Java Servlet Specification Version 2.4 ships with the Sun Java System Application Server 8.2 release, and can be downloaded from the following URL:

  http://java.sun.com/products/servlet/ (`http://java.sun.com/products/servlet/`)

  Version 2.3 of the specification shipped with the J2EE 1.3 SDK. The following items discuss compatibility issues between these releases.

  - `HttpSessionListener` `sessionDestroyed` method was previously used to notify that a session was invalidated. As of this release, this method is used to notify that a session is about to be invalidated so that it notifies before the session invalidation. If the code assumed the previous behavior, it must be modified to match the new behavior.
  - `ServletRequest`, `getRemotePort`, `getLocalName`, `getLocalAddr`, `getLocalPort`

    The following methods are added in the `ServletRequest` interface in this version of the specification. Be aware that this addition causes source incompatibility in some cases, such as when a developer implements the `ServletRequest` interface. In this case, ensure that all the new methods are implemented:

    - `public int getRemotePort()` returns the Internet Protocol (IP) source port of the client or last proxy that sent the request.
    - `public java.lang.String getLocalName()` returns the host name of the IP interface on which the request was received.
    - `public java.lang.String getLocalAddr()` returns the IP address of the interface on which the request was received.
    - `public int getLocalPort()` returns the IP port number of the interface on which the request was received.

Java Server Pages (JSP) Specification 2.0 ships with the Sun Java System Application Server 8.2 release and is downloadable from the following URL:

http://java.sun.com/products/jsp/ (`http://java.sun.com/products/jsp/`)

JSP specification 1.2 shipped with the J2EE 1.3 SDK. Wherever possible, the JSP 2.0 specification attempts to be fully backward compatible with the JSP 1.2 specification. In some cases, there are ambiguities in the JSP 1.2 specification that have been clarified in the JSP 2.0 Specification. Because some JSP 1.2 containers behave differently, some applications that rely on container-specific behavior may need to be adjusted to work correctly in a JSP 2.0 environment.

The following is a list of known backward compatibility issues related to JSP:

- Tag Library validators that are not namespace-aware and that rely solely on the prefix parameter might not correctly validate some JSP 2.0 pages. This is because the XML view might contain tag library declarations in elements other than `jsp:root`, and might contain the same tag library declaration more than once, using different prefixes. The uri parameter should always be used by tag library validators instead. Existing JSP pages with existing tag libraries do not create any problems.
- You may observe differences in I18N behavior on some containers primarily due to ambiguity in the JSP 1.2 specification. Where possible, steps were taken to minimize the impact on backward compatibility and overall, the I18N abilities of technology have been greatly improved.

  In the JSP specification versions previous to JSP 2.0, JSP pages in XML syntax and those in standard syntax determined their page encoding in the same fashion, by examining the `pageEncoding` or `contentType` attributes of their page directive, defaulting to ISO-8859-1 if neither was present.

  As of the JSP specification v2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the `pageEncoding` attribute of those pages is only checked to make sure it is consistent with the page encoding determined as per the XML specification.

  As a result of this change, JSP documents that rely on their page encoding to be determined from their `pageEncoding` attribute will no longer be decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

  Additionally, in the JSP 1.2 Specification, page encodings are determined on a per translation unit basis whereas in the JSP 2.0 Specification, page encodings are determined on a per-file basis. Therefore, if a.jsp statically includes b.jsp, and a page encoding is specified in a.jsp but not in b.jsp, in the JSP 1.2 specification a.jsp's encoding is used for b.jsp, but in the JSP 2.0 Specification, the default encoding is used for b.jsp.
- The type coercion rules (shown in Table JSP.1-11 in the JSP 2.0 specification) have been reconciled with the EL coercion rules. There are some exceptional conditions that no longer result in an exception in the JSP 2.0 specification. In particular, when passing an

empty String to an attribute of a numeric type, a translation error or a NumberFormatException used to occur, whereas in the JSP 2.0 specification, a 0 is passed in instead. See Table JSP.1-11 in the JSP 2.0 specification for details. In general, this is not expected to cause any problems because these would have been exceptional conditions in the JSP 1.2 specification and the specification allowed for these exceptions to occur at translation time or request time.

- The JSP container uses web.xml to determine the default behavior of various container features. The following is a list of items of which JSP developers should be aware when upgrading their web.xml file from Servlet version 2.3 Specification to Servlet version 2.4 Specification.
  - EL expressions are ignored by default in applications created with JSP 1.2 technology. When upgrading a Web application to the JSP 2.0 specification, EL expressions are interpreted by default. The escape sequence \\$ can be used to escape EL expressions that should not be interpreted by the container. Alternatively, the isELIgnored page directive attribute, or the el-ignored configuration element can deactivate EL for entire translation units. Users of JSTL 1.0 need to either upgrade their taglib/ imports to the JSTL 1.1 URIs, or they need to use the _rt versions of the tags (for example c_rt instead of c, or fmt_rt instead of fmt).
  - Files with an extension of .jspx are interpreted as JSP documents by default. Use the JSP configuration element is-xml to treat .jspx files as regular JSP pages. There is no way to disassociate .jspx from the JSP container.
  - The escape sequence \\$ was not reserved in the JSP 1.2 specification. Any template text or attribute value that appeared as \\$ in the JSP 1.2 specification used to output \\$ but now outputs just $.

JAXP and SAX Incompatibilities

Sun Java System Application Server 8.2 supports JAXP 1.3, which in turn supports SAX 2.0.2. In SAX 2.0.2, DeclHandler.externalEntityDecl requires the parser to return the absolute system identifier for consistency with DTDHandler.unparsedEntityDecl. This might cause some incompatibilities when migrating applications that use SAX 2.0.0.

To migrate an application that uses SAX 2.0.0 to SAX 2.0.2 without changing the previous behavior of externalEntityDecl, you can set the resolve-dtd-uris feature to false. For example:

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://xml.org/sax/features/resolve-dtd-uris",false);
```

Other incompatibilities between SAX 2.0.0 and SAX 2.0.2 are documented in the JAXP Compatibility Guide.

The `pass-by-reference` Element

Sun Java System Application Server 8.2 is compatible with the Java 2 Platform, Enterprise Edition specification by default. In this case, all portable J2EE programs run on the Application Server without modification. However, as allowed by the J2EE compatibility requirements, it is possible to configure applications to use features of the Sun Java System Application Server 8.2 that are not compatible with the J2EE specification.

The pass-by-reference element in the `sun-ejb-jar.xml` file only applies to remote calls. As defined in the EJB 2.0 specification, section 5.4, calls to local interfaces use pass-by-reference semantics.

If the pass-by-reference element is set to its default value of false, the parameter passing semantics for calls to remote interfaces comply with the EJB 2.0 specification, section 5.4. If set to true, remote calls involve pass-by-reference semantics instead of pass-by-value semantics, contrary to this specification.

Portable programs cannot assume that a copy of the object is made during such a call, and thus that it is safe to modify the original. Nor can they assume that a copy is not made, and thus that changes to the object are visible to both caller and callee. When this flag is set to true, parameters and return values are considered read-only. The behavior of a program that modifies such parameters or return values is undefined. For more information about the pass-by-reference element, see the *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

CHAPTER 6

Migrating from Application Server 6.x/7.x

This chapter describes the considerations and strategies that are needed when moving J2EE applications to the Application Server Enterprise Edition 8.2 product line.

The sections that follow describe issues that arise while migrating the main components of a typical J2EE application from Application Server 6.x/7.x to Application Server Enterprise Edition 8.2.

This chapter contains the following sections:

The migration issues described in this chapter are based on an actual migration that was performed for a J2EE application called *iBank*, a simulated online banking service, from Application Server 6.x/7.x to Sun Java System Application Server 8.2. This application reflects all aspects of a traditional J2EE application.

The following areas of the J2EE specification are covered by the iBank application:

- Servlets, especially with redirection to JSP pages (model-view-controller architecture)
- JSP pages, especially with static and dynamic inclusion of pages
- JSP custom tag libraries
- Creation and management of HTTP sessions
- Database access through the JDBC API
- Enterprise Java Beans: Stateful and Stateless session beans, CMP and BMP entity beans.
- Assembly and deployment in line with the standard packaging methods of the J2EE application

Migrating from Application Server 6.x

This section covers the following topics:

Migrating Deployment Descriptors

There are two types of deployment descriptors, namely, Standard Deployment Descriptors and Runtime Deployment Descriptors. Standard deployment descriptors are portable across J2EE platform versions and vendors and does not require any modifications. Currently, there are exceptions due to standards interpretation. The following table lists such deployment descriptors.

Source Deployment Descriptor	Target Deployment Descriptor
`ejb-jar.xml` - 1.1	`ejb-jar.xml` - 2.0
`web.xml`	`web.xml`
`application.xml`	`application.xml`

The J2EE standard deployment descriptors `ejb-jar.xml`, `web.xml` and `application.xml` are not modified significantly. However, the `ejb-jar.xml` deployment descriptor is modified to make it compliant with EJB 2.0 specification in order to make the application deployable on Sun Java System Application Server 8.2.

Runtime deployment descriptors are vendor specific and product specific and are not portable across application servers due to difference in their format. Hence, deployment descriptors require migration. This section describes how you can manually create the runtime deployment descriptors and migrate relevant information.

The following table summarizes the deployment descriptor migration mapping.

Source Deployment Descriptor	Target Deployment Descriptor
`ias-ejb-jar.xml`	`sun-ejb-jar.xml`
`<bean-name>-ias-cmp.xml`	`sun-cmp-mappings.xml`
`ias-web.xml`	`sun-web.xml`

The standard deployment descriptors of Application Server 6.x needs modification when moving to Application Server 8.2 because of nonconformance with the DTDs.

A majority of the information required for creating `sun-ejb-jar.xml` and `sun-web.xml` comes from `ias-ejb-jar.xml` and `ias-web.xml` respectively. However, there is some information that is required and extracted from the home interface (.java file) of the CMP entity bean, in case the `sun-ejb-jar.xml` being migrated declares one. This is required to build the `<query-filter>` construct inside the `sun-ejb-jar.xml`, which requires information from inside the home interface of that CMP entity bean. If the source file is not present during the migration time, the `<query-filter>` construct is created, but with missing information (which manifests itself in the form of REPLACE ME phrases in the migrated `sun-ejb-jar.xml`.

Additionally, if the `ias-ejb-jar.xml` contains a `<message-driven>` element, then information from inside this element is picked up and used to fill up information inside both `ejb-jar.xml` and `sun-ejb-jar.xml`. Also, inside the `<message-driven>` element of `ias-ejb-jar.xml`, there is an element `<destination-name>`, which holds the JNDI name of the topic or queue to which the MDB listens. In Application Server 6.5, the naming convention for this JNDI name is `cn=<SOME_NAME>.` Since a JMS Topic or Queue with this name is not deployable on Application Server, the application server changes this to `<SOME_NAME>`, and inserts this information in the `sun-ejb-jar.xml`. This change must be reflected for all valid input files, namely, all `.java`, `.jsp` and `.xml` files. Hence, this JNDI name change is propagated across the application, and if some source files that contain reference to this JNDI name are unavailable, the administrator must make the changes manually so that the application becomes deployable.

Migrating Web Applications

Application Server 6.x support servlets (Servlet API 2.2), and JSPs (JSP 1.1). Sun Java System Application Server 8.2 supports Servlet API 2.4 and JSP 2.0.

Within these environments it is essential to group the different components of an application (servlets, JSP and HTML pages and other resources) together within an archive file (J2EE-standard Web application module) deploying it on the application server.

According to the J2EE specification, a Web application is an archive file (WAR file) with the following structure:

- A root directory containing the HTML pages, JSP, images and other static resources of the application.

- A `META-INF/` directory containing the archive manifest file `MANIFEST.MF` containing the version information for the SDK used and, optionally, a list of the files contained in the archive.
- A `WEB-INF/` directory containing the application deployment descriptor (`web.xml` file) and all the Java classes and libraries used by the application, organized as follows:
  - A `classes/` sub-directory containing the tree-structure of the compiled classes of the application (servlets, auxiliary classes), organized into packages
    - A `lib/` directory containing any Java libraries (JAR files) used by the application

Migrating Java Server Pages and JSP Custom Tag Libraries

Application Server 6.x complies with the JSP 1.1 specification and Application Server 8.2 complies with the JSP 2.0 specification.

JSP 2.0 specification contains many new features, as well as updates to the JSP 1.1 specification.

These changes are enhancements and are not required to migrate to JSP pages from JSP 1.1 to 2.0.

The implementation of JSP custom tag libraries in Application Server 6.x complies with the J2EE specification. Consequently, migrating JSP custom tag libraries to the Application Server Enterprise Edition 8.2does not pose any particular problem, nor require any modifications.

Migrating Servlets

Application Server 6.x supports the Servlet 2.2 API. Sun Java System Application Server 8.2 supports the Servlet 2.4 API.

Servlet API 2.4 leaves the core of servlets relatively untouched. Most changes are concerned with adding new features outside the core.

The most significant features are:

- Servlets now require JDK 1.2 or later
- Filter mechanisms have been created
- Application lifecycle events have been added
- Internationalization support has been added
- Error and security attributes have been expanded
- `HttpUtils` class has been deprecated
- Several DTD behaviors have been expanded and clarified

These changes are enhancements and are not required to be made when migrating servlets from Servlet API 2.2 to 2.4.

However, if the servlets in the application use JNDI to access resources in the J2EE application (such as data sources or EJBs), some modifications might be needed in the source files or in the deployment descriptor.

One last scenario might require modifications to the servlet code. Naming conflicts can occur with Application Server 6.x if a JSP page has the same name as an existing Java class. In this case, the conflict must be resolved by modifying the name of the JSP page in question. This in turn can mean editing the code of the servlets that call this JSP page. This issue is resolved in Application Server as it uses a new class loader hierarchy. In the new version of the application server, for a given application, one class loader loads all EJB modules and another class loader loads web module. As these two loaders do not talk with each other, there is no naming conflict.

To obtain a reference to a data source bound to the JNDI context, look up the data source's JNDI name from the initial context object. The object retrieved in this way is then be *cast* as a `DataSource` type object:

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

For detailed information, refer to section "Migrating JDBC Code."

The actual migration of the components of a Servlet or JSP application from Application Server 6.x to Application Server 8.2does not require any modifications to the component code.

If the Web application is using a server resource, such as a data source, the Application Server requires that this resource to be declared inside the `web.xml` file and, correspondingly, inside the `sun-web.xml` file. To declare a data source called `jdbc/iBank`, the `<resource-ref>` tag in the `web.xml` file is as follows:

```
<resource-ref>
   <res-ref-name>jdbc/iBank</res-ref-name>
   <res-type>javax.sql.XADataSource</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

The corresponding declaration inside the `sun-web.xml` file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE FIX ME: need confirmation on the DTD to be used for this file
<sun-web-app>
   <resource-ref>
      <res-ref-name>jdbc/iBank</res-ref-name>
      <jndi-name>jdbc/iBank</jndi-name>
   </resource-ref>
 </sun-web-app>
```

Migrating Web Application Modules

Migrating applications from Application Server 6.x to Sun Java System Application Server 8.2 does not require any changes to the Java code or Java Server Pages. However, you must change the following files:

- web.xml
- ias-web.xml

The Application Server adheres to J2EE 1.4 standards, according to which, the web.xml file inside a WAR file must comply with the revised DTD at http://java.sun.com/dtd/web-app_2_3.dtd. This DTD is a superset of the previous versions' DTD, hence only the <! DOCTYPE definition needs to be changed inside the web.xml file, which is to be migrated. The modified <! DOCTYPE declaration looks like:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//
                        DTD Web Application 2.3//EN"
                        "http://java.sun.com/dtd/web-app_2_3.dtd">
```

In Application Server Enterprise Edition 8.2, the name of this file is changed to sun-web.xml.

This XML file must declare the Application Server-specific properties and resources that are required by the Web application.

If the ias-web.xml of the Application Server 6.5 application is present and does declare Application Server 6.5 specific properties, then this file needs to be migrated to Application Server standards. The DTD file name has to be changed to sun-web.xml. For more details, see URL http://wwws.sun.com/software/dtd/appserver/sun-web-app_2_4-1.dtd

Once you have made these changes to the web.xml and ias-web.xml files, the Web application (WAR file) can be deployed from the Application Server's deploytool GUI interface or from the command line utility asadmin. The deployment command must specific the type of application as web.

Invoke the asadmin command line utility by running asadmin.bat file or the asadmin.sh script in the Application Server's bin directory.

The command at the asadmin prompt is:

```
asadmin deploy -u username -w password
-H hostname
-p adminport
--type web
[--contextroot contextroot]
[--force=true]
[--name component-name]
[--upload=true] filepath
```

Migrating Enterprise EJB Modules

Application Server 6.x supports EJB 1.1, and the Application Server supports EJB 2.0. Therefore, both can support:

- Stateful or stateless session beans

- Entity beans with BMP or CMP

EJB 2.0, however, introduces a new type of enterprise bean, called an MDB.

J2EE 1.4 specification dictates that the different components of an EJB must be grouped together in a JAR file with the following structure:

- `META-INF/` directory with an XML deployment descriptor named `ejb-jar.xml`
- The `.class` files corresponding to the home interface, remote interface, the implementation class, and the auxiliary classes of the bean with their package

Application Server 6.x use this archive structure. However, the EJB 1.1 specification leaves each EJB container vendor to implement certain aspects as they see fit:

- Database persistence of CMP EJBs (particularly the configuration of mapping between the bean's CMP fields and columns in a database table).
- Implementation of the custom finder method logic for CMP beans.
- Application Server 6.x andApplication Server 8.2do not handle migrations in the same way, which means that some XML files must be modified:
- The `<!DOCTYPE` definition must be modified to point to the latest DTD URL (in the case of J2EE standard deployment descriptors, such as `ejb-jar.xml).`
- Replace the `ias-ejb-jar.xml` file with the modified version of this file (for example, file `sun-ejb-jar.xml`, which is created manually according to the DTDs). For more information, see http://www.sun.com/software/dtd/appserver/sun-ejb-jar_2_1-1.dtd
- Replace all the <*ejb-name*>`-ias-cmp.xml` files with one `sun-cmp-mappings.xml` file, which is created manually. For more information, see http://www.sun.com/software/dtd/appserver/sun-cmp-mapping_1_2.dtd
- Optionally, for CMP entity beans, use the `capture-schema` utility in the Application Server's `bin` directory to generate the dbschema. Then place it above the `META-INF` directory for the entity beans.

EJB Migration

As mentioned in Chapter 3, "Migrating J2EE Applications," while Application Server 6.x supports the EJB 1.1 specification, Application Server also supports the EJB 2.0 specification. The EJB 2.0 specification introduces the following new features and functions to the architecture:

- MDBs
- Improvements in CMP
- Container-managed relationships for entity beans with CMP
- Local interfaces
- EJB Query Language (EJB QL)

Although the EJB 1.1 specification continues to be supported in the Application Server, the use of the EJB 2.0 architecture is recommended to leverage its enhanced capabilities.

For detailed information on migrating from EJB 1.1 to EJB 2.0, please refer to Chapter 4, "Migrating from EJB 1.1 to EJB 2.0"

DTD Changes

Migrating EJBs from Application Server 6.x to Application Server 8.2 is done without making any changes to the EJB code. However, the following DTD changes are required.

- The `<!DOCTYPE>` definition must be modified to point to the latest DTDs with J2EE standard deployment descriptors, such as `ejb-jar.xml`.
- Replace `ias-ejb-jar.xml` file with the modified version of this file, named `sun-ejb-jar.xml`, created manually according to the deployment descriptors. For more details, see http://wwws.sun.com/software/dtd/appserver/sun-ejb-jar_2_1-1.dtd
- In the `sun-ejb-jar.xml` file, the JNDI name for all the EJBs must be added before "`ejb/`' in all the JNDI names. This is required because, in Application Server 6.5, the JNDI name of the EJB can only be `ejb/`<*ejb-name*> where <*ejb-name*> is the name of the EJB as declared inside the `ejb-jar.xml` file.

  In the Application Server, a new tag has been introduced in the `sun-ejb-jar.xml`. This is where the JNDI name of the EJB is declared.

Note – To avoid changing JNDI names throughout the application, declare the JNDI name of the EJB as `ejb/`<*ejb-name*> inside the <*jndi-name*> tag.

Migrating EJB Applications that Support SFSB Failover

Sun ONE Application Server 6.5 supports failover of stateful session beans. To take advantage of the SFSB failover in 6.5, the session bean need to be configured with failover and Distributed Store or DSync. The DSync mechanism is used to save the session beans's conversational state during runtime.

Note – Sun ONE Application Server 6.5 does not support failover of stateful session beans for rich clients on the RMI/IIOP path. Such applications can take advantage of SFSB failover on the RMI/IIOP path in Sun Java System Application Server 8.2. For more information on SFSB failover configuration, see "Stateful Session Bean Failover" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Sun Java System Application Server 8.2, Enterprise Edition supports failover of stateful session beans. Application Server 8.2 uses the High Availability Database (HADB) for storing session data. The principle followed in supporting SFSB failover in saving the conversational state of an

SFSB at predefined points in its lifecycle to a persistent store. This mechanism is referred to as *checkpointing*. In case of a server crash, the checkpointed state of an SFSB can be retrieved from the persistent store. In order to use HADB for storing session data, you must configure HADB as the persistent store. The underlying store for the HTTP sessions and stateful session beans is same and the configuration of persistent store is exactly similar to configuration of session store.

For information on configuring HADB for session failover, see Chapter 9, "Configuring High Availability Session Persistence and Failover," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Migration of stateful session beans deployed in Sun ONE Application Server 6.5 to Sun Java System Application Server 8.2 does not require any changes in the EJB code. However, the following steps must be performed:

- Modify the `<!DOCTYPE` definition to point to the latest DTD URL in case of J2EE standard deployment descriptors, such as `ejb-jar.xml`.
- Replace `ias-ejb-jar.xml` with the modified version of this file, i.e., `sun-ejb-jar.xml`, which is created manually according to the DTDs.
- Replace all the <*ejb-name*>`-ias-cmp.xml` files with one `sun-cmp-mappings.xml` file, which is created manually.
- No changes are required in the application source code for taking advantage of the SFSB state failover support. All configuration needed for checkpointing SFSBs will be applied at the Application Server specific deployment descriptor (`sun-ejb-jar.xml`), or in the domain configuration file (`domain.xml`).

  However, if you are accessing the EJBs through servlets then you need to store the EJB home and remote references in the session. The following is the code example to store `ejbHome` and `ejbRemote` interfaces in the session:

  ```
  session.setAttribute("ejbhome", ejbHome);
  session.setAttribute("ejbremote", ejbRemote);
  ```

  The following code example demonstrates how to retrieve the `ejbHome` and `ejbRemote` from the session:

  ```
  ejbHome = session.getAttribute("ejbhome");
  ejbRemote = session.getAttribute("ejbremote");
  ```
- In the `domain.xml`, make sure that the `availability-enabled` attribute of `availability-service` element is set to TRUE. If `availability-enabled` attribute is set to TRUE indicates that failover is enabled at the server instance level. That is, if a server instance fails to process a request, the request is routed to the next available server instance.

  SFSB checkpointing adds performance overhead on the EJB container, you may want to restrict checkpointing to a list of SFSBs whose state failover is critical to the application.

You can enable/disable the checkpointing at the method level in `sun-ejb-jar.xml`. For more details see “Specifying Methods to Be Checkpointed” in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

If, in the deployment descriptor for the SFSB EJB module in 6.5 (`ias-ejb-jar.xml`), the `failoverrequired` attribute of the `session` element is set to TRUE, you might want to enable availability-service for such EJB modules in the Application Server 8.2 environment.

Entity Beans

- The `<!DOCTYPE>` definition must be modified to point to the latest DTDs containing J2EE standard deployment descriptors, such as `ejb-jar.xml`.
- Update the `<cmp-version>` tag with the value 1.1, for all CMPs in the `ejb-jar.xml` file.
- Replace all the <*ejb-name*>`-ias-cmp.xml` files with the manually created `sun-cmp-mappings.xml` file. For more information, see http://wwws.sun.com/software/dtd/appserver/sun-cmp-mapping_1_2.dtd
- Generate `dbschema` by using the `capture-schema` utility in the Application Server installation’s bin directory and place it above `META-INF` folder for Entity beans.
- `Replace` the `ias-ejb-jar.xml` with the `sun-ejb.jar.xml` in Application Server.
- In Application Server 6.5, the finder's SQL was directly embedded into the <*ejb-name*>`-ias-cmp.xml`. In Application Server, mathematical expressions are used to declare the `<query-filter>` for the various finder methods.

Message Driven Beans

Application Server provides seamless Message Driven Support through the tight integration of Sun Java System Message Queue with the Application Server, providing a native, built-in JMS Service.

This installation provides Application Server with a JMS messaging system that supports any number of Application Server instances. Each server instance, by default, has an associated built-in JMS Service that supports all JMS clients running in the instance.

Both container-managed and bean-managed transactions, as defined in the Enterprise JavaBeans Specification, v2.0, are supported.

Message Driven Bean support in iPlanet Application Server was restricted to developers, and used many of the older proprietary APIs. Messaging services were provided by iPlanet Message Queue for Java 2.0. An LDAP directory was also required under iPlanet Application Server to configure the `Queue Connection Factory` object.

The `QueueConnectionFactory`, and other elements required to configure Message Driven Beans in Application Server are now specified in the `ejb-jar.xml` file.

For more information on the changes to deployment descriptors, see "Migrating Enterprise Applications" on page 77 For information on Message Driven Beans see "Using Message-Driven Beans" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

Migrating Enterprise Applications

According to the J2EE specifications, an enterprise application is an EAR file, which must have the following structure:

- A `META-INF/` directory containing the XML deployment descriptor of the J2EE application called `application.xml`
- The JAR and WAR archive files for the EJB modules and Web module of the enterprise application, respectively

In the application deployment descriptor, the modules that make up the enterprise application and the Web application's context root are defined.

Application server 6.x and the Application Server 8.2support the J2EE model wherein applications are packaged in the form of an enterprise archive (EAR) file (extension `.ear`). The application is further subdivided into a collection of J2EE modules, packaged into Java archives (JAR files, which have a `.jar` file extension) and EJBs and Web archives (WAR files, which have a `.war` file extension) for servlets and JSPs.

It is essential to follow the steps listed here before deploying an enterprise application:

▼ To Build an EAR File

1 **Package EJBs in one or more EJB modules.**

2 **Package the components of the Web application in a Web module.**

3 **Assemble the EJB modules and Web modules in an enterprise application module.**

4 **Define the name of the enterprise application's root context, which will determine the URL for accessing the application.**

The Application Server uses a newer class loader hierarchy than Application Server 6.x does. In the new scheme, for a given application, one class loader loads all EJB modules and another class loader loads Web modules. These two are related in a parent child hierarchy where the JAR module class loader is the parent module of the WAR module class loader. All classes loaded by the JAR class loader are available/accessible to the WAR module but the reverse is not true. If a certain class is required by the JAR file as well as the WAR file, then the class file must be packaged inside the JAR module only. If this guideline is not followed it can lead to class conflicts.

Application Root Context and Access URL

There is a major difference between Application Server 6.x and the Application Server8.2, concerning the applications access URL (root context of the application's Web module). If `AppName` is the name of the root context of an application deployed on a server called `hostname`, the access URL for this application differs, depending on the application server used:

- With Application Server 6.x, which is always used jointly with a Web front-end, the access URL for the application takes the following form (assuming the Web server is configured on the standard HTTP port, 80):

  ```
  http://<hostname>/NASApp/AppName/
  ```
- With the Application Server8.2, the URL takes the form:

  ```
  http://<hostname>:<portnumber>/AppName/
  ```

The TCP port used as default by Application Server 8.2is port 8080.

Although the difference in access URLs between Application Server 6.x and the Application Server might appear minor, it can be problematic when migrating applications that make use of absolute URL references. In such cases, it is necessary to edit the code to update any absolute URL references so that they are no longer prefixed with the specific marker used by the Web Server plug-in for Application Server 6.x.

Applications With Form-based Authentication

Applications developed on Application Server 6.5 that use form-based authentication can pass the request parameters to the Authentication Form or the Login page. The Login page could be customized to display the authentication parameters based on the input parameters.

For example:

```
http://gatekeeper.uk.sun.com:8690/NASApp/test/secured/page.jsp?
arg1=test&arg2=m
```

Application Server 8.2 does not support the passing of request parameters while displaying the Login page. The applications that uses form-based authentication, which passes the request parameters can not be migrated to Application Server8.2. Porting such applications to Application Server8.2 requires significant changes in the code. Instead, you can store the request parameter information in the session, which can be retrieved while displaying the Login page.

The following code example demonstrates the workaround:

Before changing the code in 6.5:

```
---------index-65.jsp -----------
<%@page contentType="text/html"%>
<html>
```

```
<head><title>JSP Page</title></head>
<body>
go to the <a href="secured/page.htm">secured a rea</a>
</body>
</html>
----------login-65.jsp--------------
<%@page contentType="text/html"%>
<html>
<head> </head>
<body>
<!-- Print login form -->
<h3>Parameters</h3><br>
out.println("arg1 is " + request.getParameter("arg1"));
out.println("arg2 is " + request.getParameter("arg2"));
</body>
</html>
```

After changing the code in Application Server8.2:

```
---------index-81.jsp -----------
<%@page contentType="text/html"%>
<html>
<head><title>JSP Page</title></head>
<body>
<%session.setAttribute("arg1","test"); %>
<%session.setAttribute("arg2","me"); %>
go to the <a href="secured/page.htm">secured area</a>
</body>
</html>
```

The `index-81.jsp` shows how you can store the request parameters in a session.

```
----------login-81.jsp--------------
<%@page contentType="text/html"%>
<html>
<head> </head>
<body>
<!-- Print login form -->
<h3>Parameters</h3><br>
<!--retrieving the parameters from the session -->
out.println("arg1 is"+(String)session.getAttribute("arg1"));
out.println("arg2 is” + (String)session.getAttribute("arg2"));
</body>
</html>
```

Migrating Proprietary Extensions

A number of classes proprietary to the Application Server 6.x environment might have been used in applications. Some of the proprietary packages used by Application Server 6.x are listed below:

- `com.iplanet.server.servlet.extension`
- `com.kivasoft.dlm`
- `com.iplanetiplanet.server.jdbc`
- `com.kivasoft.util`
- `com.netscape.server.servlet.extension`
- `com.kivasoft`
- `com.netscape.server`

These APIs are not supported in the Application Server8.2. Applications using any classes belonging to the above package must be rewritten to use standard J2EE APIs. Applications using custom JSP tags and UIF framework also need to be rewritten to use standard J2EE APIs.

Migrating UIF

The Application Server 8.2does not support the use of Unified Integration Framework (UIF) API for applications. Instead, it supports the use of J2EE Connector Architecture (JCA) for integrating the applications. However, the applications developed in Application Server 6.5 use the UIF. In order to deploy such applications to the Application Server8.2, migrate the UIF to the J2EE Connector Architecture. This section discusses the prerequisites and steps to migrate the applications using UIF to Application Server.

Before migrating the applications, ensure that the UIF is installed on Application Server 6.5. To check for the installation, use the following approaches:

- Checking in the Registry Files
- Checking for UIF Binaries in Installation Directories

Checking in the Registry Files

UIF is installed as a set of application server extensions. They are registered in the application server registry during the installation. Search for the following strings in the registry to check whether UIF is installed.

Extension Name Set:

- Extension DataObjectExt-cDataObject
- Extension RepositoryExt-cLDAPRepository
- Extension MetadataService-cMetadataService
- Extension RepoValidator-cRepoValidator
- Extension BSPRuntime-cBSPRuntime

- Extension BSPErrorLogExt-cErrorLogMgr
- Extension BSPUserMap-cBSPUserMap

The registry file on Solaris Operating Environment can be found at the following location:

AS_HOME`/AS/registry/reg.dat`

Checking for UIF Binaries in Installation Directories

UIF installers copy specific binary files in to the application server installation. Successfully finding the files listed below, indicates that UIF is installed.

The location of the following files on Solaris and Windows is:

AS_HOME`/AS/APPS/bin`

List of files to be searched on Solaris:

- `libcBSPRlop.so`
- `libcBSPRuntime.so`
- `libcBSPUserMap.so`
- `libcDataObject.so`
- `libcErrorLogMgr.so`
- `libcLDAPRepository.so`
- `libcMetadataService.so`
- `libcRepoValidator.so`
- `libjx2cBSPRuntime.so`
- `libjx2cDataObject.so`
- `libjx2cLDAPRepository.so`
- `libjx2cMetadataService.so`

List of files to be searched on Windows:

- `cBSPRlop.dll`
- `cBSPRuntime.dll`
- `cBSPUserMap.dll`
- `cDataObject.dll`
- `ErrorLogMgr.dll`
- `cLDAPRepository.dll`
- `cMetadataService.dll`
- `cRepoValidator.dll`
- `jx2cBSPRuntime.dll`
- `jx2cDataObject.dll`
- `jx2cLDAPRepository.dll`
- `jx2cMetadataService.dll`

Before migrating the UIF to Application Server8.2, ensure that the UIF API is being used in the applications. To verify its usage:

- Check for the usage of `netscape.bsp` package name in the Java sources
- Check for the usage of `access_cBSPRuntime.getcBSPRuntime` method in the sources. You must call this method to acquire the UIF runtime.

Contact appserver-migration@sun.com for information about UIF migration to the Application Server8.2.

Migrating JDBC Code

With the JDBC API, there are two methods of database access:

- Establishing Connections Through the `DriverManager` Interface

  (JDBC 1.0 API), by loading a specific driver and providing a connection URL. This method is used by other Application Servers, such as IBM's WebSphere 4.0
- Using JDBC 2.0 Data Sources

  The `DataSource` interface (JDBC 2.0 API) can be used via a configurable connection pool. According to J2EE 1.2, a data source is accessed through the JNDI naming service

---

Note – Application Server8.2 does not support the Native Type 2 JDBC drivers bundled with Application Server 6.x. Code that uses the Type 2 drivers to access third party JDBC drivers, must be manually migrated.

Establishing Connections Through the DriverManager Interface

Although this database access method is not recommended, as it is obsolete and is not very effective, there could be some applications that still use this approach.

In this case, the access code is similar to the following:

```
public static final String driver = "oracle.jdbc.driver.OracleDriver";
public static final String url =
  "jdbc:oracle:thin:tmb_user/tmb_user@iben:1521:tmbank";
Class.forName(driver).newInstance();
Properties props = new Properties();
props.setProperty("user", "tmb_user");
props.setProperty("password", "tmb_user");
Connection conn = DriverManager.getConnection(url, props);
```

This code can be fully ported from Application Server 6.x to Application Server8.2, as long as the Application Server8.2 is able to locate the classes needed to load the right JDBC driver. To make the required classes accessible to the application deployed in the Application Server, place the archive (JAR or ZIP) for the driver implementation in the `/lib` directory of the Application Server installation directory.

Modify the *CLASSPATH* by setting the path for the driver through the Admin Console GUI.

- Click the server instance `server1`.
- Click the JVM Settings tab from the right pane.
- Click the Path Settings option and add the path in the classpath suffix text entry box.
- Once the changes are made, click Save.
- Apply the new settings.
- Restart the server to modify the configuration file, `server.xml`.

Using JDBC 2.0 data sources to access a database provides performance advantages, such as transparent connection pooling, enhanced productivity by simplifying code and implementation, and code portability.

If there is a data source by the name `xyz` on Application Server 6.x application and you do not want any impact on your JNDI lookup code, make sure that the data source you create for Application Server8.2 is prefixed with JDBC. For example: `jdbc/xyz`.

For information on configuring JDBC data sources, see Chapter 3, "JDBC Resources," in *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

▼ To Connect to a Data Source

1 **Obtain the initial JNDI context.**

To guarantee portability between different environments, the code used to retrieve an `InitialContext` object (in a servlet, in a JSP page, or an EJB) is as follows:

```
InitialContext ctx = new InitialContext();
```

2 **Use a JNDI lookup to obtain a data source reference.**

To obtain a reference to a data source bound to the JNDI context, look up the data source's JNDI name from the initial context object. The object retrieved in this way is cast as a `DataSource` type object:

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

3 **Use the data source reference to obtain the connection.**

This operation requires the following line of code:

```
conn = ds.getConnection();
```

Application Server 6.x and Application Server both follow these technique to obtain a connection from the data source.

Migrating Rich Clients

This section describes the steps for migrating RMI/IIOP and ACC clients developed in Planet Application Server 6.x to the Application Server8.2.

Authenticating a Client

Application Server 6.x provides a client-side callback mechanism that enables applications to collect authentication data from the user, such as the username and the password. The authentication data collected by the iPlanet CORBA infrastructure is propagated to the application server via IIOP.

If ORBIX 2000 is the ORB used for RMI/IIOP, portable interceptors implement security by providing hooks, or interception points, which define stages within the request and reply sequence.

In Application Server 8.2, The authentication is done based on JAAS (Java Authorization and Authentication System API). If a client does not provide a `CallbackHandler`, then the default `CallbackHandler`, called the `LoginModule`, is used by the ACC to obtain the authentication data.

For detailed instructions on using JAAS for authentication, see Chapter 9, "Configuring Security," in *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

Using ACC

In Application Server 6.x, no separate `appclient` script is provided. You are required to place the `iasacc.jar` file in the classpath instead of the `iascleint.jar` file. The only benefit of using the Application Client Container (ACC) for packaging application clients in 6.x is that the JNDI names specified in the client application are indirectly mapped to the absolute JNDI names of the EJBs.

In case of Application Server 6.x applications, a standalone client uses the absolute name of the EJB in the JNDI lookup. That is, outside an ACC, the following approach is used to lookup the JNDI:

```
initial.lookup(“ejb/ejb-name”);
initial.lookup(“ejb/module-name/ejb-name”);
```

If your application was developed using Application Server 6.5 SP3, you would have used the prefix `java:comp/env/ejb/` when performing lookups by using absolute references.

```
initial.lookup("java:comp/env/ejb/ejb-name");
```

In Sun Java System Application Server 8.2, the JNDI lookup is done on the `jndi-name` of the EJB. The absolute name of the EJB must not be used. Also, the prefix, `java:comp/env/ejb` is not supported in Sun Java System Application Server 8.2. Replace the `iasclient.jar`, `iasacc.jar`, or `javax.jar` JAR files in the classpath with `appserv-ext.jar`.

If your application provides load balancing capabilities, in Sun Java System Application Server 8.2, load balancing capabilities are supported only in the form of `S1ASCTXFactory` as the context factory on the client side and then specifying the alternate hosts and ports in the cluster by setting the `com.sun.appserv.iiop.loadbalancingpolicy` system property as follows:

```
com.sun.appserv.iiop.loadbalancingpolicy=

roundrobin,host1:port1,host2:port2,...,
```

This property provides the administrator with a list of host:port combinations to round robin the ORBs. These host names can also map to multiple IP addresses. If this property is used along with `org.omg.CORBA.ORBInitialHost` and `org.omg.CORBA.ORBInitialPort` as system properties, the round robin algorithm will round robin across all the values provided. If, however, a host name and port number are provided in your code, in the environment object, that value overrides any other system property settings.

The Provider URL to which the client is connected in Application Server 6.5 is the IIOP host and port of the CORBA Executive Engine (CXS Engine). In case of Sun Java System Application Server 8.2, the client needs to specify the IIOP listener Host and Port number of the instance. No separate CXS engine exists in Sun Java System Application Server 8.2.

The default IIOP port is 3700 in Sun Java System Application Server 8.2; the actual value of the IIOP Port can be found in the `domain.xml` configuration file.

Load balancing is handled implicitly by the CXS engine in Sun ONE Application Server 6.5 upon number of Java engines registered. In Application Server 8.2 Enterprise Edition, this feature requires explicit configuration details from the clients.

After migrating the deployment descriptors from 6.x to 8.2, provide the configuration details in the `sun-acc.xml` file to enable failover capabilities in your ACC client. See "Migrating Enterprise Applications" on page 77 for information on migrating deployment descriptors.

Define the load balancing properties in the `sun-acc.xml` file to provide a highly available ACC client. The properties are defined as property elements in the `sun-acc.xml` file.

- `com.sun.appserv.iiop.endpoints`

  This property defines the list of one or more IIOP endpoints. An endpoint is specified as *host*:*port* where host is the name or IP address of the system where Application Server 8.2 is running. Port is the IIOP port at which the server is listening for IIOP requests.
- `com.sun.appserv.iiop.loadbalancingpolicy`

  If the endpoint property is specified, then, this property is used to specify the load balancing policy. The value for this property must be InitialContext-based.

For example:

```
<client-container>
   <target-server name="qasol-e1" address="qasol-e1" port="3700">
   <property name="com.sun.appserv.iiop.loadbalancingpolicy"
             value="ic-based" />
   <property name="com.sun.appserv.iiop.endpoints"
             value="qasol-e1:3700,qasol-e1:3800" />
</client-container>
```

To failover an ACC client on the RMI/IIOP path, information about all the endpoints in a cluster to which the RMI/ IIOP requests can be failed over must be available. You must have defined the IIOP endpoints in the `domain.xml` file. The `iiop-cluster` element under the `availability-service` element defines the IIOP endpoints.

For more information, see Chapter 5, "Configuring HTTP Load Balancing," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Migrating Applications to Support HTTP Failover

Application Server, Enterprise Edition 8.2 supports load balancing and HTTP session persistence. The primary goal of loadbalancing is to distribute the work load between multiple server instances, thereby increasing overall throughput of the system.

For information on configuring HTTP session failover, see "HTTP Session Failover" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

To migrate 6.x HTTP applications to Application Server 8.2 EE environment and enable load-balancing capabilities, perform the following steps. Note that, no code changes will be required in the application.

▼ To Migrate and Enable Loadbalancing

1. **Make sure that at least two application server instances are created and configured.**

2. **Rename the `ias-web-app.xml` to `sun-web.xml`.**

   For more information on migrating the deployment descriptors, see the "Migrating Enterprise Applications" on page 77.

3. **Update the <DOCTYPE definition with the following code:**

   ```
   <!DOCTYPE web-app PUBLIC
   '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
   'http://java.sun.com/j2ee/dtds/web-app_2_3-1.dtd'>
   ```

4 In Sun ONE Application Server 6.5, the failover of HTTP applications was based on Dsync mechanism. The configuration for HTTP failover was done in the `ias-web-app.xml` file.

The `<server-info>` element defined under the `<servlet-info>` element, specifies whether the server on which the servlet will be served from is enabled.

The `<session-info>` element defines the following:

- `dsync-type`: This can take the value `dsync-distributed` or `dsync-local`.

  `dsync-distributed` implies that the session is distributed and thus available on all configured servers.

  `dsync-local` implies that the session is available on available only on the server on which the session was created.

  - `impl`: This can take the values `distributed` or `lite`.

    `distributed` implies that the session on distributed.

    `lite` implies that the session is local to the Java engine where the session was created. If this value is set, the dsync-type setting is ignored.

    In Sun Java System Application Server 8.2, to enable failover of applications on the HTTP route, you define the following properties in the sun-specific web application deployment descriptor file: `sun-web.xml.`
  - `persistence-store` - This can take the values memory, file, or ha. In 6.5, however, only memory based persistence store was supported.
  - `persistence-scope` - define the scope of persistence.
    - `session` - For every session, the session information will be saved.
    - `modified-session` - Only the modified session data will be stored.
    - `modified-attribute` - Only the modified attribute data will be stored. In 6.5, only modified-attribute scope was supported.

    `persistenceFrequency` - The frequency can be for every web method or time based. In 6.5, only web-method was supported.

    - `web-method` - The session state is stored at the end of each web request prior to sending a response back to the client. This mode provides the best guarantee that the session state is fully updated in case of failure.
    - `time-based` - The session state is stored in the background at the specified frequency. This mode provides less of a guarantee that the session state is fully updated. However, it can provide a significant performance improvement because the state is not stored after each request.

      A sample of the `sun-web.xml` file is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE
sun-web-app PUBLIC ’-//Sun Microsystems, Inc.//
```

```
            DTD Sun ONE Application Server 7.1 Servlet 2.3//EN’
"http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-1.dtd’>
  <sun-web-app>
    <session-config>
      <seession-manager>
        <manager-properties>
          <property name="persistence-type" value "ha’>
          <property name="persistenceFrequency" value ="web-based">
        </manager-properties>
        <store-properties>
          <property name="persistenceScope" value="session">
        </store-properties>
      </session-manager>
    </session-config>
</sun-web-app>
```

For more information on the `sun-web.xml` configuration file, see “The sun-web.xml File” in *Sun Java System Application Server Enterprise Edition 8.2 Developer’s Guide*.

5 **Sun Java System Application Server 8.2 requires the load balancer plug-in to be installed and configured, in order to loadbalance the HTTP request and failover the requests to available server instances in a cluster when there is a failure.**

For more information about the load balancer, see Chapter 5, “Configuring HTTP Load Balancing,” in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

6 **In the `load-balancer.xml` file, make sure that the `web-module` `enabled` element is set to true.**

```
<loadbalancer>
  <cluster name=cluster1>
  ...
    <web-module context-root="abc" enabled=true>
  </cluster>
  <property name="https-routing" value="true"/>
</loadbalancer>
```

`enabled=true` specifies that the web module is active (enabled) for requests to be load balanced to it.

7 **Define the `https-routing` property and set its value to `true`.**

For more information on editing the `load-balancer.xml` file, see Chapter 5, “Configuring HTTP Load Balancing,” in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Deploy the applications on all server instances that is participating in load balancing.

Migrating Applications from Application Server 7

This sections contains the following topics:

- "Migrating Rich Clients" on page 89
- "Migrating EJB Applications to Support SFSB Failover" on page 91

Migrating Rich Clients

Migrating rich clients that are deployed in Application Server 7 PE/SE to Application Server 8.2 is rather simple. The deployment descriptors used in Application Server 7 can be used as is in Application Server 8.2. However, if you wish to enable loadbalancing and failover features in your client applications, you need to configure the loadbalancing and failover capabilities in the deployment descriptors.

▼ To Migrate Rich Clients from 7 PE/SE to 8.2 EE

1 **Identify the components which were installed previously.**

2 **Find out the server-instances, using `asadmin` command or through the directory listing.**

The `asadmin` command requires administration instances to be running. However, administration instances need not be running if the directory listing is used to identify the instances.

3 **In the `server.xml` file, add the following jvm-options under `jvm-config` element to enable RMI/IIOP failover feature:**

```
<jvm-config java-home=path...server-classpath=path>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
com.sun.appserv.ee.iiop.EEORBInitializer
  </jvm-option>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
com.sun.appserv.ee.iiop.EEIORInterceptorInitializer
  </jvm-option>
  <jvm-option>
    Dcom.sun.CORBA.connection.ORBSocketFactoryClass=
com.sun.appserv.enterprise.iiop.EEIIOPSocketFactory
  </jvm-option>
</jvm-config>
```

4 **Update the `availability-service` element with `availability-enabled` flag set to True:**

```
<availability-service availability-enabled="true">
  <persistence-store>
```

```
    <property-name="store-pool-jndi-name" value="" />
    <property-name="cluster-id" value="cluster1" />
  </persistence-store>
</availability-service>
```

5 **Modify the server classpath entry under the `java-config` element to include:**

install_dir/`SUNWhads/4.2.2-17/lib/hadbjdbc.jar;`

install_dir/`lib/appserv-rt-ee.jar`

6 **Add the following `jvm-option` under the `java-config` element:**

```
<jvm-option>
```

```
Dcom.sun.aas.hadbRoot=install-dir/SUNWhadb/4.2.2-17

</jvm-option>
```

7 **Update the `sun-acc.xml` with the following new load-balancing properties:**

```
<property-name="com.sun.appserv.iiop.loadbalancingpolicy"
                    value="ic-based" />
<property name="com.sun.appserv.iiop.endpoints" value=<host>:<port>" />
```

▼ To Migrate Rich Clients From 7 EE to 8.2 EE

1 **Add the following `jvm-option` elements under the `java-config` element for enabling the RMI/IIOP failover feature. (To make the class names of the following `jvm-option` elements fit on the page, they have been split in two and carried to the next line. When adding them to your project, do not split them in two as they are here.)**

```
<jvm-config java-home=path...server-classpath=path>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
          com.sun.appserv.ee.iiop.EEORBInitializer
  </jvm-option>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
          com.sun.appserv.ee.iiop.EEIORInterceptorInitializer
  </jvm-option>
  <jvm-option>
    Dcom.sun.CORBA.connection.ORBSocketFactoryClass=
          com.sun.appserv.enterprise.iiop.EEIIOPSocketFactory
  </jvm-option>
</jvm-config>
```

2 **Add the following entry in `server.xml` to setup the iiop-cluster.**

```
<iiop-cluster>
     <iiop-server-instance name=<server-name>>
```

```
        <iiop-endpoint id=orb-listener-id,
                       host=hostname,
                       port=orb-listener-port/>
    </iiop-server-instance>
</iiop-cluster>
```

3 **Update `sun-acc.xml` with the following new entries:**

```
<property-name=ÂÂ[a8]com.sun.appserv.iiop.loadbalancingpolicy"
                    value="ic-based" />
<property name="com.sun.appserv.iiop.endpoints"
          value="hostname:port" />
```

Migrating EJB Applications to Support SFSB Failover

Application Server 7 does not support failover of Stateful Session Beans (SFSB). Application Server Enterprise Edition 8.2 supports failover of stateful session beans on the HTTP and RMI/IIOP path. This section describes the procedure to migrate EJB applications from Application Server 7 SE/PE/EE to Application Server 8.2 EE to support SFSB state failover.

Migrating EJB Applications From 7 SE/PE/EE to 8.2 EE

To achieve high availability of EJB applications that use stateful session beans to persist the data, you need to configure a persistent store for each cluster of application servers, where client session information can be maintained across potential failures of individual appserver instances. In addition, the availability-enabled flag must be turned on for each server instance in the cluster.

Application Server 8.2 EE supports the failover of stateful session beans. In order to enable this feature in your EJB applications that were deployed to Application Server 8.2 EE, follow the steps below:

To migrate Entity beans from previous releases of Sun's Application Server, follow the procedure described in "Entity Beans" on page 76.

SFSB failover is supported when the SFSB is accessed from EJBs, servlets, or Java Server Pages in applications executing in the same application server process. The SFSB can be accessed through either a local or remote interface.

To take advantage of SFSB state failover support, you need not edit the code. However, you need to provide all the configuration parameters needed for checkpointing the SFSBs in the Sun-specific deployment descriptor (`sun-ejb-jar.xml`) or in the server configuration file.

For detailed information on SFSB failover, see "Stateful Session Bean Failover" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Index