Sun Java System Application Server 9.1 Upgrade and Migration Guide

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

Part No: 819–3676–11
February 2009

090204@21808

Contents

Preface

This guide explains how to upgrade and migrate Java™ applications from the Sun Java System Application Server 8.x to the Sun Java SystemApplication Server 9.1 product line. This guide also explains how to migrate Java applications from Sun ONE Application Server 6.x/7 (also known as iPlanet Application Server), Java Enterprise Edition (Java EE™) Reference Implementation (RI) 1.3 Application Server, Sun Java System Application Server 8.x, WebLogic Application Server, WebSphere Application Server, JBoss, and so on to Application Server 9.1.

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 Application Server documentation is `http://docs.sun.com/coll/1343.4`. 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
Documentation Center	Application Server documentation topics organized by task and subject.
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™), and database drivers.
Quick Start Guide	How to get started with the Application Server product.
Installation Guide	Installing the software and its components.
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.
Application Deployment Guide	Deployment of applications and application components to the Application Server. Includes information about deployment descriptors.

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

Book Title	Description
Developer's Guide	Creating and implementing Java Platform, Enterprise Edition (Java EE platform) applications intended to run on the Application Server that follow the open Java standards model for Java EE components and APIs. Includes information about developer tools, security, debugging, and creating lifecycle modules.
Java EE 5 Tutorial	Using Java EE 5 platform technologies and APIs to develop Java EE applications.
Java WSIT Tutorial	Developing web applications using the Web Service Interoperability Technologies (WSIT). Describes how, when, and why to use the WSIT technologies and the features and options that each technology supports.
Administration Guide	System administration for the Application Server, including configuration, monitoring, security, resource management, and web services management.
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	Upgrading from an older version of Application Server or migrating Java EE applications from competitive application servers. 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 Sun Java Enterprise System (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 documentation about other stand-alone Sun Java System server products, go to the following:

- Message Queue documentation (`http://docs.sun.com/coll/1343.4`)
- Directory Server documentation (`http://docs.sun.com/coll/1224.1`)
- Web Server documentation (`http://docs.sun.com/coll/1308.3`)

A Javadoc™ tool reference for packages provided with the Application Server is located at `http://glassfish.dev.java.net/nonav/javaee5/api/index.html`. Additionally, the following resources might be useful:

- The Java EE 5 Specifications (`http://java.sun.com/javaee/5/javatech.html`)
- The Java EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

For information on creating enterprise applications in the NetBeans™ Integrated Development Environment (IDE), see `http://www.netbeans.org/kb/55/index.html`.

For information about the Java DB database included with the Application Server, see `http://developers.sun.com/javadb/`.

The GlassFish Samples project is a collection of sample applications that demonstrate a broad range of Java EE technologies. The GlassFish Samples are bundled with the Java EE Software Development Kit (SDK), and are also available from the GlassFish Samples project page at `https://glassfish-samples.dev.java.net/`.

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

as-install	Represents the base installation directory for Application Server.	Java ES installations on the Solaris™ operating system:
		`/opt/SUNWappserver/appserver`
		Java ES installations on the Linux operating system:
		`/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`

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

Placeholder	Description	Default Value

domain-root-dir	Represents the directory containing all domains.	Java ES Solaris installations:
		`/var/opt/SUNWappserver/domains/`
		Java ES Linux installations:
		`/var/opt/sun/appserver/domains/`
		All other installations:
		as-install`/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*
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.
-	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℠ 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 819-3676.

◆ ◆ ◆ CHAPTER 1

Application Server Compatibility Issues

Application Server 9.1 is binary compatible with Application Server 8.2, 8.1, 8.0, and 7.x. Java applications that run on versions 8.1, 8.0, and 7.x also work on Application Server 9.1 except for the incompatibilities listed in this chapter.

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

Application Client Interoperability

Application clients use EJBs, web services, or other enterprise components that are in the application server (on the server side). The application client and the application server must use the same version and implementation of the RMI-IIOP protocol. Application Server 9.1 does not support communication between different versions of the protocol implementation. You cannot run application clients with one version of the application server runtime with a server that has a different version. Most often, this would happen if you upgraded the server but had not upgraded all the application client installations.

You can use the Java Web Start support to distribute and launch the application client. If the runtime on the server has changed since the end-user last used the application client, Java Web Start automatically retrieves the updated runtime. Java Web Start enables you to keep the clients and servers synchronized and using the same runtime.

HTTP File Caching

HTTP file caching, which was present in Application Server 8, has been discontinued in Application Server 9.1.

Default Admin Port

The default admin port in Application Server 7 was 4848. The default port in Application Server 8.x was 4849. In Application Server 9.1, the default port is 4848.

`domain.xml` Elements

If you have not configured message-level security providers for a server instance, Application Server 8 applies default configurations from the Domain Administration Server (DAS). Application Server 9.1 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 9.1, the Upgrade tool does not add these missing elements in the `domain.xml` file.

Deprecated Attributes

The `anonymous-role` attribute is present in the DTD but the use of this attribute is deprecated. This attribute has been removed from the template that generates `domain.xml`. The `forced-response-type` and `default-response-type` attributes are deprecated. Use `forced-type` and `default-type` instead.

System Properties

The default security policy of Application Server 9.1 does not allow you to change some system properties. For example, in Application Server 8, the read/write permission of `java.util.PropertyPermission` property is `"*", "read,write";`. In Application Server 9.1 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 9.1, 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 9.1:

- `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 8, the tag data resides in the `server.xml` and for in Application Server 9.1, in `domain.xml`.

The upgrade tool locates the class file defined for custom realms and makes it available to the Application Server 9.1 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 8 to `com.sun.enterprise.security.auth.realm` in Application Server 9.1. 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;
```

Sun Deployment Descriptor: `sun-web.xml`

In Application Server 8, the default value for the optional attribute delegate was false. In Application Server 9.1, 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 8, the default value of the `encodeCookies` property was true. This property was not present in Application Server 8. In Application Server 9.1, 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 8.

CORBA Performance Option

In Application Server 8, 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 9.1. 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 9.1, `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 9.1 can be used to move from `server.xml`in Application Server 8 to `domain.xml` in Application Server 9.1.

The `lib/dtds/sun-domain_1_1.dtd` file for Application Server 9.1 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 8 are not supported in Application Server 9.1. In Application Server 9.1, 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 9.1 Reference Manual*.

Primary Key Attribute Values

In Application Server 8, it was possible to change any field (in the Admin Console) or attribute (in the Command Line Interface (CLI)). In Application Server 9.1, 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 9.1 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 9.1 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 9.1, 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 9.1 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 9.1:

- `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 9.1. The software license key and web core were removed, and Application Server 9.1 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 8, 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 9.1, 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 8 are not supported in Application Server 9.1 .

In Application Server 9.1, 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 9.1, see the *Sun Java System Application Server 9.1 Administration Guide*.

This table displays a one-to-one mapping of the incompatibilities in dotted names between Application Server 8 and 9.1. 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 9.1 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 9.1 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 9.1 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 9.1 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 9.1, attributes with no values contain empty strings, as they did in Application Server 8.

◆ ◆ ◆ CHAPTER 2

Upgrading an Application Server Installation

The Upgrade tool, which is bundled with Application Server 9.1, 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 9.1. 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	9.1
7.X PE	Not supported
7.XSE	Not supported
7.XEE	Not supported

TABLE 2–1 Supported Upgrade Paths *(Continued)*

Source Installation	9.1
8.0PE	Supported (Upgrade from 8.0 PE domain to 9.1 developer domain is supported)
8.1PE	Supported (Upgrade from 8.1 PE domain to 9.1 developer domain is supported)
8.1EE	Supported (Upgrade from 8.1 EE domain to 9.1 enterprise domain is supported)
8.2PE	Supported (Upgrade from 8.2 PE domain to 9.1 developer domain is supported)
8.2EE	Supported (Upgrade from 8.2 EE domain to 9.1 enterprise domain is supported)
9.0PE	Supported (Upgrade from 9.0 PE domain to 9.1 developer domain is supported)

Note – Only the enterprise profile supports upgrades from Application Server Enterprise Edition 8.x.

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.

Note – Ensure that the `-c/--console` option is the first option in the command line, if you want to run `asupgrade` in CLI mode.

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 the `asenv.conf` file (on Solaris) or the `asenv.bat` file (on Windows).
- 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 you stop all server instances, node agents, and domains (in that order) in the source server (the server from which you are upgrading) and the target server (the server to which you are upgrading).

Migration of Deployed Applications

Application archives (EAR files) and component archives (JAR, WAR, and RAR files) that are deployed in the Application Server 8.x environment do not require any modification to run on Application Server 9.1.

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.

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

In Application Server 8.x, the clusters are defined in the `domain.xml` file and there is no need to specify clusters separately. 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.

Transfer of Certificates and Realm Files

The Upgrade tool transfers certificates from the source certificate database to the target. The tool transfers security policies, password files from standard, file-based realms, and custom realm classes.

Upgrade Verification

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.

After you have upgrade a domain, you can see a file whose name is in the following format: `upgradedTo`<*releasenumber*>. For example, a domain that has been upgrade to 9.1 will have a file called `upgradeTo91` in its `config` folder.

Upgrade Rollback

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

Note – You can cancel the upgrade process only if you are running the Upgrade Tool in GUI mode.

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.
- Inline Upgrade: You can use the `start-domain` command to upgrade domains of Application Server 8.x or 9.0 to Application Server 9.1. This type of upgrade works only if you are performing an in-place upgrade of binaries.

Upgrading from Java ES 5 or Java ES 5 Update 1

This section provides the following procedures :

- "To Upgrade to Application Server 9.1 (Java ES Update 1)" on page 31
- "To Upgrade Service Registry" on page 32

▼ To Upgrade to Application Server 9.1 (Java ES Update 1)

If you have Application Server 8.x installed as part of Java ES 5 or Java ES 5 Update 1, you can use the following upgrade procedure to upgrade to Application Server 9.1, which is distributed with Java ES 5 Update 1 as an optional download.

Java ES 5 users can use the following procedure to upgrade from Application Server 8.x Enterprise Edition to Application Server 9.1

1. **Stop all instances, node agents, and domains running on Application Server 8.x.**
2. **Start the Application Server 9.1 installer . For instructions on how to install Application Server 9.1, see "Installing Application Server 9.1" in** *Sun Java System Application Server 9.1 Installation Guide***.**

3 **For Solaris or Linux, choose the same installation directory as that of the Application Server 8.x installation. For Windows, choose a different installation directory and not the Application Server 8.x installation directory.**

The installer updates the required shared components. The installer also created a new domain (domain1) in *as-install*`/appserver/domains/domain1`on Solaris, *as-install*`/domains/domain1` on Linux, and *as-install*`\domains\domain1` on Windows.

4 **Start the Upgrade tool. When prompted for source , provide the 8.x domain directory. When prompted for the target, provide the 9.1 domains root directory.**

This tool is located in the *as-install*`/appserver/bin` directory on Solaris, *as-install*`/bin` directory on Linux, and *as-install*`\bin` directory on Windows.

See "Upgrading your Application Server" on page 33 for the syntax and usage of the Upgrade tool or `asupgrade` command.

Note – Application Server 9.1 is not supported with Java ES 5 and needs the Java ES 5 Update 1 Portal Server. Portal Server for Java ES 5 Update 1 is not supported on Windows.

▼ To Upgrade Service Registry

Java ES users can use this procedure to upgrade Service Registry domains.

1 **Upgrade Service Registry.**

See the Java ES Upgrade guide on docs.sun.com for instructions.

2 **Install Application Server 9.1. For instructions on how to install Application Server 9.1, see "Installing Application Server 9.1" in** *Sun Java System Application Server 9.1 Installation Guide***.**

3 **Run the following commands:**

`cd` <*service_registry_install_dir*>`/install`

On Solaris, <*service_registry_install_dir*> is `/opt/SUNWsrvc-registry`, by default. On Linux, the default install location is `/opt/sun/srvc-registry`.

<*Ant-base_dir*>`/ant -f build-install.xml appserver.deploy.as9`

Where <*Ant-base_dir*> is `/usr/sfw/bin` on Solaris and `/opt/sun/share/bin` on Linux.

4 **Run the Upgrade Tool to upgrade the Service Registry domain.**

Provide <*service_registry_domainbase*>`/domains/registry` as input for the Source Server field (`--source` option).

Provide <*Application_Server9.1_install_dir*>`/domains` as input for the Target Server field (`--target` option).

For instructions on how to run the Upgrade Tool, see "Upgrading your Application Server" on page 33.

Upgrading your Application Server

For upgrading your Application Server installation you can choose:

- "To Upgrade from the Command Line" on page 33
- "To Upgrade using the Upgrade Tool Wizard" on page 34

Use the following procedures to upgrade your existing clusters and node agents:

- "To Upgrade a Cluster" on page 34
- "To Upgrade a Node Agent" on page 35

To Upgrade from the Command Line

You can run the upgrade utility from the command line using the following syntax:

```
asupgrade
[--console ]
[--version ]
[--help ]
[--source applicationserver_8.x_installation_domaindirectory]
[--target applicationserver_9.1_installation]
[--passwordfile passwords.txt]
```

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 or -V	--version	The version of the Upgrade Tool.
-h	--help	Displays the arguments for launching the upgrade utility.
-t	--target	The domains directory of the Application Server 9.1 installation.
-s	--source	The installation directory of the older Application Server installation.
-a	--adminuser	The admin user for the source server.
-f	--passwordfile	The file containing the admin password and the master password.

Note – For a more detailed usage summary of the asupgrade command, see `asupgrade(1M)`.

The following examples show how to use the `asupgrade` command-line utility to upgrade an existing application server installation to Application Server 9.1.

This example shows how to perform a side-by-side upgrade of a Sun Java SystemApplication Server 8.x installation to Sun Java System Application Server 9.1.

```
asupgrade --source /home/sunas8.2/domains/domain1 --target /home/sjsas9.1/domains
```

After upgrade, node agents for all remote instances are created on the target DAS. These node agents have to be copied to the respective host systems and started.

▼ To Upgrade using the Upgrade Tool Wizard

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.

1. **In the Source Installation Directory field, enter the location of the existing installation from which to import the configuration. Enter the domain directory.**
   For example, <install-root>/domains/domain1
2. **In the Target Installation Directory field, enter the location of the Application Server installation to which to transfer the configuration. Provide the domains root directory of the target Application Server installation as the input to this field.**
3. **Provide the admin user name, the admin password, and master password of the source application server. The target domain is created with these credentials.**
4. **The Upgrade Results panel is displayed showing the status of the upgrade operation.**
5. **Click the Finish button to close the Upgrade Tool when the upgrade process is complete.**

To Upgrade a Cluster

When you are upgrading Application Server 8.x EE to Application Server 9.1, the upgrade tool automatically detects clusters, if any, on the source installation.

▼ To Upgrade a Node Agent

If you are performing an upgrade from Application Server 8.x EE to Application Server 9.1, in which all the node agents run on a single machine, the upgrade tool automatically detects node agents, if any, on the source installation. The user need not take any special action. If you are performing an upgrade from Application Server 8.x EE to Application Server 9.1, in which remote node agents are running on other machines use the following steps to perform the upgrade.

1 **Install Application Server 9.1 on** `Machine A`**.**

2 **Perform the upgrade from Application Server 8.x EE to Application Server 9.1.**

3 **Install Application Server 9.1 on** `Machine B` **without the DAS but with the Node Agent feature.**

Note – `Machine A` is the primary machine. It runs the DAS. `Machine B` is a secondary machine, which is not running the DAS. `Machine B` runs remote node agents that are configured to communicate with `Machine A`.

---

4 **If you are performing an in-place upgrade:**

   a. **On** `Machine A`**, start each node agent using the** `start-node-agent` **command with the** `--syncinstances` **option. This option resynchronizes all associated instances. Example:**
   `asadmin start-node-agent --user admin --syncinstances nodeagent1`

   b. **On** `Machine B`**, start each node agent using the** `start-node-agent` **command with the** `--syncinstances` **option. This option resynchronizes all associated instances**

5 **If you are performing an side—by—side upgrade:**

   a. **Check the value of the** `agent.adminPort` **property in the**`nodeagent.properties` **file before starting the node agent for the first time. Perform this check on the** `nodeagent.properties` **files on both** `Machine A` **and** `Machine B`**. The value of** `agent.adminPort` **property must reflect the same value as the** `jmx-connector` **port defined in the** `domain.xml` **file on** `Machine A`**. Edit the** `agent.adminPort` **property in the** `nodeagent.properties` **files on** `Machine A` **and** `Machine B`**, as required.**

   b. **If you are using non-default ports, you must check the value of the** `agent.bind.status` **property in** `nodeagent.properties` **file on** `Machine B`**, before starting the node agent for the first time. If the** `agent.bind.status` **property in** `nodeagent.properties` **file is** `BOUND`**, change it to** `UNBOUND`**.**

   c. **On** `Machine A`**, start each node agent using the** `start-node-agent` **command. Do not use the** `--syncinstances` **option.**

d. On `Machine B`, start each node agent using the `start-node-agent` command. Do not use the `--syncinstances` option.

More Information Starting the Upgraded Node Agent

For information on how to resolve problems with starting the upgraded node agent, see "Node Agent Startup Failure" on page 36.

Correcting Potential Upgrade Problems

This section addresses the following issues that could occur during an upgrade to Application Server 9.1:

Node Agent Startup Failure

The default admin port in Application Server 9.1 is 4848 and in Application Server 8.x EE, the default admin port is 4849. When you upgrade from Application Server 8.x EE, you could run into problem while trying to start the default node agent that exists in the target, due to the port clash.

To resolve this problem, edit the `das.properties` file before starting the target domain or node agent. Change the `agent.das.port` property to the admin port value in the upgraded `domain.xml`, which is 4849.

Upgrade tool leaves the node agent in a rendezvous false state. If the `agent.bind.status` property in `nodeagent.properties` file is `BOUND`, change it to `UNBOUND`. The node agent starts up successfully after making these changes.

`_TimerPool` Issue

The datasource class used for a `jdbc-connection-pool` resource named `__TimerPool` has changed from `org.apache.derby.jdbc.EmbeddedXADataSource` in Application Server 8.x EE to `org.apache.derby.jdbc.ClientDataSource` in Application Server 9.1. This change requires a addition of two property elements, User and Password to the `jdbc-connection-pool` element in the `domain.xml` file. Edit the Application Server 9.1 `domain.xml` file and add the appropriate user name and password. Example:

```
<property name="User" value="APP"/> <property name="Password" value="APP"/>
```

Problems Due to Missing Client JAR Files

You have deployed applications that use client JARs in Application Server 8.x. You upgrade your existing installation to Application Server 9.1. You could run into problems while trying to run these applications (that were deployed in Application Server 8.x) in Application Server 9.1.

To solve this problem, perform the following steps:

1. After upgrade, start Application Server 9.1.
2. Use the `asadmin get-client-stubs` command to transfer the missing client stubs to a local directory. See `get-client-stubs(1)`.
3. Run the appclient pointing to the client JAR files in the local directory.

Problems with Migrated Applications that Use JavaDB

You have deployed applications that use JavaDB databases in Application Server 8.x. You upgrade your existing installation to Application Server 9.1. You run the `asadmin start-database` command and successfully start JavaDB. In this scenario, you could run into problems while trying to run these applications (that were deployed in Application Server 8.x) in Application Server 9.1 because the instance directory of JavaDB in Application Server 9.1 has changed.

To solve this problem, perform the following steps:

1. After upgrade, start Application Server 9.1.
2. Use the `asadmin start-database` command with `--dbhome` option pointing to older (Application Server 8.x) version of JavaDB. Example `asadmin start-database --dbhome /home/johnsmith/appserver8.2/databases`
3. Deploy the migrated applications.

`classpath-suffix` and `classpath-prefix` Not Transferred

Upgrade tool does not transfer `java-config` attributes, `classpath-suffix`, `classpath-prefix`, `java-home`, `server-classpath`, because the information provided can be implementation specific.

JVM Options Not Transferred

When you upgrade from a previous version of the application server, transfer of the previous configuration is required. Since the target configuration files may have new parameters and new preconfigured features, copying the old configuration files to the new server installation is not possible. The values of the old configurations must be transferred to the Application Server 9.1 configuration format.

The following JVM options are not transferred from the source to the target installation:

- `Dorg.xml.sax.driver`
- `Dcom.sun.jdo.api.persistence.model.multipleClassLoaders`
- `Djava.util.logging.manager`
- `Dcom.sun.aas.imqLib`
- `Dcom.sun.aas.imqBin`
- `Dcom.sun.aas.webServicesLib`
- `Dcom.sun.aas.configRoot 8. Xmx<...>m`

The options that are not transferred are listed down in the upgrade log. The user can manually change such attributes in the configuration file.

Port Conflict Problems

After upgrading the source server to Application Server 9.1, start the domain and then the node agent, which, by default, starts the server instances. If you have upgraded from Application Server 8.x EE, you might face problems while attempting to start the node agent. The domain, clusters, and instances have admin port set to 4849 and the node agent points to 4848. You need to manually modify the admin port to which the node agent points. To change the node agent port, edit the `agent.das.port` property in the *install_dir*`/nodeagents/`*node-agent-name*`/`*server_name*`/config/das.properties` file.

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.

Note – The default ports in Application Server 9.1 are:

- 4848 for admin port
- 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

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 8uses 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.

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

While upgrading from Application Server 8.x EE to Application Server 9.1, during a side-by-side upgrade, you will not be able to point your new 9.1 load balancer plug-in to the old 8.x 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 9.1 load balancer plug-in installation to the instance belonging to the new installation.

▼ Migration of Additional HTTP Listeners Defined on the Source Server to the Target Server

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.**

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.**

▼ Migration of Additional HTTP and IIOP Listeners Defined on the Source Server to the Target 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.**

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 Java EE 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

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

The following topics are addressed:

Java EE Components and Standards

Sun Java System Application Server 9.1 (hereafter called Application Server) is a Java EE-compliant server based on the component standards developed by the Java community. By contrast, the Sun Java SystemApplication Server 7 (Application Server 8) 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 four versions, there are considerable differences with the 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 9.1, 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 Java EE Components

Component API	Sun ONE Application Server 6.x	Sun Java System Application Server 7	Sun Java System Application Server 8.2	Sun Java System Application Server 9.1
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	

Java EE Application Components

Java EE 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 Java EE components include:

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

Java EE components are packaged separately and bundled into a Java EE 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 Java EE

application. A Java EE application is composed of one or more enterprise bean(s), Web, or application client component modules. The final enterprise solution can use one Java EE application or be made up of two or more Java EE applications, depending on design requirements.

A Java EE 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 Java EE 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 Java EE, see Java EE website:

Why is Migration Necessary?

Although Java EE 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 Java EE 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 Java EE application components, modules, and files deployed on other established application server platforms. Such differences require mappings between earlier implementation details of the Java EE 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 Java EE 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 Java EE components or applications. The Java EE 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

Java EE 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 Java EE 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 Java EE is an evolving standard, different products can support different versions of Java EE component APIs.

Migration Tool and Other Resources

The Migration Tool for Sun Java System Application Server 9.1 (hereafter called Migration Tool) migrates Java EE applications from other server platforms to Sun Java System Application Server 9.1.

The following source platforms are supported for Sun Java System Application Server 9.1:

- Sun ONE Application Server 6.*x*
- Sun Java System Application Server 7
- Sun Java System Application Server 8.0/8.1
- Java EE 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 Java EE applications to Sun Java System Application Server 9.1, 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/`).

Deploying the Migrated Application

To be able to deploy your migrated applications on Application Server 9.1, it is important to understand classloaders in Application Server 9.1 and changes to the architecture.

Application Server 8.x does not support overriding of libraries such as the default parser. Application Server 9.1 provides the `-libraries` option for overriding the default XML parser and using a different JAXP compatible implementation of the parser.

You can use the `-libraries` option to control the scope of your libraries. See the Chapter 2, "Class Loaders," in *Sun Java System Application Server 9.1 Developer's Guide* for a detailed description of the classloaderhandling mechanism in Application Server 9.1. For more on classloader delegation in Application Server 9.1, see "Class Loader Delegation" in *Sun Java System Application Server 9.1 Developer's Guide*.

◆ ◆ ◆ 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 9.1, 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

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

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 Java EE 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

In Sun Java System Application Server 9.1, 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 9.1 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 9.1.

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 9.1. 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 9.1 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 9.1. Nonetheless, it is preferable to stay as close as possible to the Java EE 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 54 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 9.1.

▼ 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 9.1.

Migration of ejb-jar.xml

To migrate the file `ejb-jar.xml` to Sun Java System Application Server 9.1, 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 9.1 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>
```

Index

F

H

I

J

M

O

P

S

T

U