Sun Java System Application Server Platform Edition 8.2 Developer's Guide

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

Part No: 819–4721–13
June 2008

080619@20490

Contents

Figures

Tables

Examples

Preface

This *Developer's Guide* describes how to create and run Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications that follow the open Java standards model for J2EE components and APIs in the Sun Java System Application Server environment. Topics include developer tools, security, assembly, deployment, debugging, and creating lifecycle modules.

Who Should Use This Book

This *Developer's Guide* is intended for use by software developers who create, assemble, and deploy J2EE applications using Sun Java System servers and software. Application Server software developers should already understand the following technologies:

- Java technology
- The Java 2 Platform, Enterprise Edition (J2EE platform), version 1.4
- Hypertext Transfer Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Extensible Markup Language (XML)

How This Book Is Organized

The *Developer's Guide* has three parts and an Appendix:

- Part I includes general development topics relevant to the Application Server, such as security and debugging.
- Part II describes J2EE application components, such as servlets and message-driven beans, that can run on the Application Server.
- Part III describes services and APIs that provide Application Server resources, such as JDBC and JNDI.
- Appendix A, "Deployment Descriptor Files," describes deployment descriptor files specific to the Application Server.

The following table summarizes the chapters in this book.

TABLE P–1 How This Book Is Organized

Chapter	Description
Chapter 1, "Setting Up a Development Environment"	Describes setting up an application development environment in the Application Server.
Chapter 2, "Securing Applications"	Explains how to write secure J2EE applications, which contain components that perform user authentication and access authorization.
Chapter 3, "Assembling and Deploying Applications"	Describes Application Server modules and how these modules are assembled separately or together in an application. Also describes class loaders and tools for assembly and deployment.
Chapter 4, "Debugging Applications"	Provides guidelines for debugging applications in the Application Server.
Chapter 5, "Developing Web Applications"	Describes how web applications are supported in the Application Server.
Chapter 6, "Using Enterprise JavaBeans Technology"	Describes how Enterprise JavaBeans™ (EJB™) technology is supported in the Application Server.
Chapter 7, "Using Container-Managed Persistence for Entity Beans"	Provides information on how container-managed persistence (CMP) works in the Application Server.
Chapter 8, "Developing Java Clients"	Describes how to develop, assemble, and deploy J2EE Application Clients.
Chapter 9, "Developing Connectors"	Describes Application Server support for the J2EE Connector 1.5 architecture.
Chapter 10, "Developing Lifecycle Listeners"	Describes how to create and use a lifecycle listener module.
Chapter 11, "Using the JDBC API for Database Access"	Explains how to use the Java Database Connectivity (JDBC™) API for database access with the Application Server.
Chapter 12, "Using the Transaction Service"	Describes J2EE transactions and transaction support in the Application Server.
Chapter 13, "Using the Java Naming and Directory Interface"	Explains how to use the Java Naming and Directory Interface™ (JNDI) API for naming and references.
Chapter 14, "Using the Java Message Service"	Explains how to use the Java Message Service (JMS) API, and describes the Application Server's fully integrated JMS provider: the Sun Java System Message Queue software.
Chapter 15, "Using the JavaMail API"	Explains how to use the JavaMail™ API.
Chapter 16, "Using the Java Management Extensions (JMX) API"	Explains how to use the Java Management Extensions (JMX™) API.
Appendix A, "Deployment Descriptor Files"	Describes deployment descriptor files specific to the Application Server.

Application Server Documentation Set

The Application Server documentation set describes deployment planning and system installation. The URL for stand-alone Application Server documentation is `http://docs.sun.com/app/docs/coll/1343.2`. For an introduction to Application Server, refer to the books in the order in which they are listed in the following table.

TABLE P–2 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, JDK, and JDBC/RDBMS.
Quick Start Guide	How to get started with the Application Server product.
Installation Guide	Installing the software and its components.
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.
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.
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 Books

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

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

The URL for all documentation about Java ES and its components is `http://docs.sun.com/prod/entsys.5`.

You can find a directory of URLs for the official specifications at *install-dir*`/docs/index.htm`. Additionally, the following resources might be useful.

General J2EE Information:

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`

Core J2EE Patterns: Best Practices and Design Strategies by Deepak Alur, John Crupi, & Dan Malks, Prentice Hall Publishing

Java Security, by Scott Oaks, O'Reilly Publishing

Programming with Servlets and JSP files:

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

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

Programming with EJB components:

Enterprise JavaBeans, by Richard Monson-Haefel, O'Reilly Publishing

Programming with JDBC:

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

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

Javadocs:

Javadocs for packages provided with the Application Server are located in *install-dir*`/docs/api`.

Default Paths and File Names

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

TABLE P–3 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™ 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*

Typographic Conventions

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

TABLE P–4 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–5 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[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 819-4721.

PART I

Developing and Deploying Applications

◆ ◆ ◆ CHAPTER 1

Setting Up a Development Environment

This chapter gives guidelines for setting up an application development environment in the Sun Java™ System Application Server. Setting up an environment for creating, assembling, deploying, and debugging your code involves installing the mainstream version of the Application Server and making use of development tools. In addition, sample applications are available. These topics are covered in the following sections:

Installing and Preparing the Server for Development

For the Sun Java Enterprise System, Application Server installation is part of the system installation process. For more information, see `http://www.sun.com/software/javaenterprisesystem/index.html`.

For all other installations, the following components are included in the full installation. For more information, see the *Sun Java System Application Server Platform Edition 8.2 Installation Guide*.

- Application Server core, including:
  - J2EE 1.4 compliant application server
  - Administration Console
  - `asadmin` utility
  - deploytool
  - Other development and deployment tools
  - Sun Java System Message Queue software
  - J2SE 1.4.2
  - Derby database
- JDK

- Sample Applications

The NetBeans IDE bundles the Platform Edition of the Application Server, so information about this IDE is provided as well.

After you have installed Application Server, you can further optimize the server for development in these ways:

- Locate utility classes and libraries so they can be accessed by the proper classloaders. For more information, see "Using the System Classloader" on page 82 or "Using the Common Classloader" on page 82.
- Set up debugging. For more information, see Chapter 4, "Debugging Applications."
- Configure the Java Virtual Machine (JVM™) software. For more information, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Tools

The following general tools are provided with the Application Server:

- "The asadmin Command" on page 42
- "The Administration Console" on page 43

The following development tools are provided with the Application Server or downloadable from Sun:

- "NetBeans IDE" on page 43
- "The asant Utility" on page 43
- "deploytool" on page 43
- "Verifier" on page 43
- "Migration Tool" on page 44

The following third-party tools might also be useful:

- "Debugging Tools" on page 44
- "Profiling Tools" on page 44

The asadmin Command

The `asadmin` command allows you to configure a local or remote server and perform both administrative and development tasks at the command line. For general information about `asadmin`, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

The `asadmin` command is located in the *install-dir*`/bin` directory. Type `asadmin help` for a list of subcommands.

The Administration Console

The Administration Console lets you configure the server and perform both administrative and development tasks using a web browser. For general information about the Administration Console, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

To access the Administration Console, type `http://host:4848` in your browser. The *host* is the name of the machine on which the Application Server is running.

NetBeans IDE

The NetBeans™ IDE (integrated development environment) allows you to create, assemble, and debug code from a single, easy-to-use interface. The Platform Edition of the Application Server is bundled with the NetBeans 5 IDE. For more information about using the NetBeans IDE, see `http://www.netbeans.org`.

The asant Utility

Apache Ant 1.6.5 is provided with the Application Server and can be launched from the `bin` directory using the command `asant`. The Application Server also provides server-specific tasks for deployment; see "asant Assembly and Deployment Tool" on page 98. The sample applications provided with the Application Server use Ant `build.xml` files; see "Sample Applications" on page 44.

For more information about Ant, see the Apache Software Foundation web site at `http://ant.apache.org/`.

deploytool

You can use the deploytool, provided with Application Server, to assemble J2EE applications and modules, configure deployment parameters, perform simple static checks, and deploy the final result. For more information about using the deploytool, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`.

Verifier

The verifier tool checks a J2EE application file (EAR, JAR, WAR, RAR), including Java classes and deployment descriptors, for compliance with J2EE specifications. Use it to check whether an application has obvious bugs and to make applications portable across application servers. The verifier can be launched from the deploytool or from the command line. For more information, see "The Deployment Descriptor Verifier" on page 84.

Migration Tool

The Migration Tool reassembles J2EE applications and modules developed on other application servers. For more information and to download the Migration Tool, see `http://java.sun.com/j2ee/tools/migration/index.html`.

For additional information on migration, see the *Sun Java System Application Server Platform Edition 8.2 Upgrade and Migration Guide*.

Debugging Tools

You can use several debuggers with the Application Server. For more information, see Chapter 4, "Debugging Applications."

Profiling Tools

You can use several profilers with the Application Server. For more information, see "Profiling" on page 122.

Sample Applications

Sample applications that you can examine and deploy are included with the full installation of the Application Server. You can also download these samples separately if you installed the Application Server without them initially.

If installed with the Application Server, the samples are in the *install-dir*`/samples` directory. The samples are organized in categories such as `ejb`, `jdbc`, `connectors`, `i18n`, and so on. Each sample category is further divided into subcategories. For example, under the `ejb` category are `stateless`, `stateful`, `security`, `mdb`, `bmp`, and `cmp` subcategories.

Most Application Server samples have the following directory structure:

- The `docs` directory contains instructions for how to use the sample.
- The `build.xml` file defines `asant` targets for the sample (see "asant Assembly and Deployment Tool" on page 98.
- The `build` and `javadocs` directories are generated as a result of targets specified in the `build.xml` file.
- The `src/java` directory under each component contains source code for the sample.
- The `src/conf` directory under each component contains the deployment descriptors.

With a few exceptions, sample applications follow the standard directory structure described here: `http://java.sun.com/blueprints/code/projectconventions.html`.

The *install-dir*`/samples/common-ant.xml` file defines properties common to all sample applications and implements targets needed to compile, assemble, deploy and undeploy sample applications. In most sample applications, the `build.xml` file includes `common-ant.xml`.

For a detailed description of the `helloworld` sample and how to deploy and run it, see the associated documentation at:

install-dir`/samples/ejb/stateless/apps/simple/docs/index.html`

After you deploy the `helloworld` sample in Application Server, you can invoke it using the following URL:

`http://`*server*`:`*port*`/helloworld`

◆ ◆ ◆ CHAPTER 2

Securing Applications

This chapter describes how to write secure J2EE applications, which contain components that perform user authentication and access authorization for servlets and EJB business logic. For information about administrative security for the server, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

This chapter contains the following sections:

Security Goals

In an enterprise computing environment, there are many security risks. The goal of the Sun Java System Application Server is to provide highly secure, interoperable, and distributed component computing based on the J2EE security model. Security goals include:

- Full compliance with the J2EE security model (for more information, see the J2EE specification, v1.4 Chapter 3 Security).
- Full compliance with the EJB v2.1 security model (for more information, see the Enterprise JavaBean specification v2.1 Chapter 15 Security Management). This includes EJB role-based authorization.

- Full compliance with the Java Servlet v2.4 security model (for more information, see the Java Servlet specification, v2.4 Chapter 11 Security). This includes servlet role-based authorization.
- Support for single sign-on across all Application Server applications within a single security domain.
- Support for message security.
- Security support for application clients.
- Support for several underlying authentication realms, such as simple file and LDAP. Certificate authentication is also supported for SSL client authentication. For Solaris, OS platform authentication is supported in addition to these.
- Support for declarative security through Application Server specific XML-based role mapping.
- Support for JACC (Java Authorization Contract for Containers) pluggable authorization as included in the J2EE 1.4 specification and defined by JSR-115.

Application Server Specific Security Features

The Application Server supports the J2EE v1.4 security model, as well as the following features which are specific to the Application Server:

- Message security; see "Configuring Message Security" on page 56
- Single sign-on across all Application Server applications within a single security domain; see "User Authentication for Single Sign-on" on page 66
- Programmatic login; see "Programmatic Login" on page 63
- A GUI-based deploytool for building XML files containing the security information; see "deploytool" on page 43

Container Security

The component containers are responsible for providing J2EE application security. There are two security forms provided by the container:

- "Programmatic Security" on page 48
- "Declarative Security" on page 49

Programmatic Security

Programmatic security is when an EJB component or servlet uses method calls to the security API, as specified by the J2EE security model, to make business logic decisions based on the

caller or remote user's security role. Programmatic security should only be used when declarative security alone is insufficient to meet the application's security model.

The J2EE specification, v1.4 defines programmatic security as consisting of two methods of the EJB `EJBContext` interface and two methods of the servlet `HttpServletRequest` interface. The Application Server supports these interfaces as specified in the specification.

For more information on programmatic security, see the following:

- Section 3.3.6, Programmatic Security, in the J2EE Specification, v1.4
- "Programmatic Login" on page 63

Declarative Security

Declarative security means that the security mechanism for an application is declared and handled externally to the application. Deployment descriptors describe the J2EE application's security structure, including security roles, access control, and authentication requirements.

The Application Server supports the deployment descriptors specified by J2EE v1.4 and has additional security elements included in its own deployment descriptors. Declarative security is the application deployer's responsibility.

There are two levels of declarative security, as follows:

- "Application Level Security" on page 49
- "Component Level Security" on page 49

Application Level Security

The application XML deployment descriptor (`application.xml`) contains descriptors for all user roles for accessing the application's servlets and EJB components. On the application level, all roles used by any application container must be listed in a `role-name` element in this file. The role names are scoped to the EJB XML deployment descriptors (`ejb-jar.xml` and `sun-ejb-jar.xml` files) and to the servlet XML deployment descriptors (`web.xml` and `sun-web.xml` files). The `sun-application.xml` file must also contain matching `security-role-mapping` elements for each `role-name` used by the application.

Component Level Security

Component level security encompasses web components and EJB components.

A secure web container authenticates users and authorizes access to a servlet or JSP by using the security policy laid out in the servlet XML deployment descriptors (`web.xml` and `sun-web.xml` files).

The EJB container is responsible for authorizing access to a bean method by using the security policy laid out in the EJB XML deployment descriptors (`ejb-jar.xml` and `sun-ejb-jar.xml` files).

Realm Configuration

This section covers the following topics:

Supported Realms

The following realms are supported in the Application Server:

- `file` - Stores user information in a file. This is the default realm when you first install the Application Server.
- `ldap` - Stores user information in an LDAP database.
- `certificate` - Sets up the user identity in the Application Server security context, and populates it with user data obtained from cryptographically verified client certificates.
- `solaris` - Allows authentication using Solaris `username+password` data. This realm is only supported on Solaris 9 and above.

For detailed information about configuring each of these realms, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

How to Configure a Realm

You can configure a realm in one of these ways:

- In the Administration Console, open the Security component under the relevant configuration and go to the Realms page. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-auth-realm` command to configure realms on local servers. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

How to Set a Realm for an Application or Module

The following deployment descriptor elements have optional `realm` or `realm-name` data subelements or attributes that override the domain's default realm:

- `sun-application` element in `sun-application.xml`
- `web-app` element in `web.xml`
- `as-context` element in `sun-ejb-jar.xml`

- `client-container` element in `sun-acc.xml`
- `client-credential` element in `sun-acc.xml`

If modules within an application specify realms, these are ignored. If present, the realm defined in `sun-application.xml` is used, otherwise the domain's default realm is used.

For example, a realm is specified in `sun-application.xml` as follows:

```
<sun-application>
    ...
    <realm>ldap</realm>
</sun-application>
```

For more information about the deployment descriptor files and elements, see Appendix A, "Deployment Descriptor Files."

Creating a Custom Realm

You can create a custom realm by providing a custom Java Authentication and Authorization Service (JAAS) login module class and a custom realm class. Note that client-side JAAS login modules are not suitable for use with the Application Server.

JAAS is a set of APIs that enable services to authenticate and enforce access controls upon users. JAAS provides a pluggable and extensible framework for programmatic user authentication and authorization. JAAS is a core API and is an underlying technology for Java EE security mechanisms. For more information about JAAS, refer to the JAAS specification for Java SDK, available at `http://java.sun.com/products/jaas/`.

For general information about realms and login modules, see the *Security* chapter of the J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`).

Custom login modules must extend the `com.sun.appserv.security.AppservPasswordLoginModule` class. This class implements `javax.security.auth.spi.LoginModule`. Custom login modules must not implement `LoginModule` directly.

Custom login modules must provide an implementation for one abstract method defined in `AppservPasswordLoginModule`:

```
abstract protected void authenticateUser() throws LoginException
```

This method performs the actual authentication. The custom login module must not implement any of the other methods, such as `login()`, `logout()`, `abort()`, `commit()`, or `initialize()`. Default implementations are provided in `AppservPasswordLoginModule` which hook into the Application Server infrastructure.

The custom login module can access the following protected object fields, which it inherits from `AppservPasswordLoginModule`. These contain the user name and password of the user to be authenticated:

```
protected String _username;
protected String _password;
```

The `authenticateUser()` method must end with the following sequence:

```
String[] grpList;
// populate grpList with the set of groups to which
// _username belongs in this realm, if any
return commitUserAuthentication(_username, _password,
  _currentRealm, grpList);
```

Custom realms must extend the `com.sun.appserv.security.AppservRealm` class and implement the following methods:

```
public void init(Properties props) throws BadRealmException,
NoSuchRealmException
```

This method is invoked during server startup when the realm is initially loaded. The `props` argument contains the properties defined for this realm in `domain.xml`. The realm can do any initialization it needs in this method. If the method returns without throwing an exception, the Application Server assumes the realm is ready to service authentication requests. If an exception is thrown, the realm is disabled.

```
public String getAuthType()
```

This method returns a descriptive string representing the type of authentication done by this realm.

```
public abstract Enumeration getGroupNames(String username) throws
InvalidOperationException, NoSuchUserException
```

This method returns an `Enumeration` (of `String` objects) enumerating the groups (if any) to which the given `username` belongs in this realm.

JACC Support

JACC (Java Authorization Contract for Containers) is part of the J2EE 1.4 specification and defined by JSR-115. JACC defines an interface for pluggable authorization providers. This provides third parties with a mechanism to develop and plug in modules that are responsible for answering authorization decisions during J2EE application execution. The interfaces and rules used for developing JACC providers are defined in the JACC 1.0 specification.

The Application Server provides a simple file-based JACC-compliant authorization engine as a default JACC provider. To configure an alternate provider using the Administration Console, open the Security component under the relevant configuration, and select the JACC Providers component. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Pluggable Audit Module Support

You can create a custom audit module. This section covers the following topics:

Configuring an Audit Module

To configure an audit module, you can perform one of the following tasks:

- To specify an audit module using the Administration Console, open the Security component under the relevant configuration, and select the Audit Modules component. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- You can use the `asadmin create-audit-module` command to configure an audit module. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

The AuditModule Class

You can create a custom audit module by implementing a class that extends `com.sun.appserv.security.AuditModule`. The `AuditModule` class provides default "no-op" implementations for each of the following methods, which your custom class can override.

```
public void init(Properties props)
```

This method is invoked during server startup when the audit module is initially loaded. The `props` argument contains the properties defined for this module in `domain.xml`. The module can do any initialization it needs in this method. If the method returns without throwing an exception, the Application Server assumes the module realm is ready to service audit requests. If an exception is thrown the module is disabled.

```
public void authentication(String user, String realm, boolean success)
```

This method is invoked when an authentication request has been processed by a realm for the given user. The `success` flag indicates whether the authorization was granted or denied.

```
public void webInvocation(String user, HttpServletRequest req, String type,
boolean success)
```

This method is invoked when a web container call has been processed by authorization. The `success` flag indicates whether the authorization was granted or denied. The `req` object is the standard `HttpServletRequest` object for this request. The `type` string is one of `hasUserDataPermission` or `hasResourcePermission` (see JSR-115).

```
public void ejbInvocation(String user, String ejb, String method, boolean
success)
```

This method is invoked when an EJB container call has been processed by authorization. The `success` flag indicates whether the authorization was granted or denied. The `ejb` and `method` strings describe the EJB component and its method that is being invoked.

The server.policy File

Each Application Server domain has its own standard J2SE policy file, located in *domain-dir*`/config`. The file is named `server.policy`.

The Application Server is a J2EE 1.4 compliant application server. As such, it follows the requirements of the J2EE specification, including the presence of the security manager (the Java component that enforces the policy) and a limited permission set for J2EE application code.

This section covers the following topics:

Default Permissions

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

Application permissions are granted in the default grant block. These permissions apply to all code not part of the internal server code listed previously. The Application Server does not distinguish between EJB and web module permissions. All code is granted the minimal set of web component permissions (which is a superset of the EJB minimal set).

A few permissions above the minimal set are also granted in the default `server.policy` file. These are necessary due to various internal dependencies of the server implementation. J2EE application developers must not rely on these additional permissions.

One additional permission is granted specifically for using connectors. If connectors are not used in a particular domain, you should remove this permission, because it is not otherwise necessary.

Changing Permissions for an Application

The default policy for each domain limits the permissions of J2EE deployed applications to the minimal set of permissions required for these applications to operate correctly. Do not add extra permissions to the default set (the grant block with no codebase, which applies to all code). Instead, add a new grant block with a codebase specific to the applications requiring the extra permissions, and only add the minimally necessary permissions in that block.

If you develop multiple applications that require more than this default set of permissions, you can add the custom permissions that your applications need. The `com.sun.aas.instanceRoot` variable refers to the `domain-dir`. For example:

```
grant "file:${com.sun.aas.instanceRoot}/applications/j2ee-apps/-" { ... }
```

You can add permissions to stub code with the following grant block:

```
grant "file:${com.sun.aas.instanceRoot}/generated/-" { ... }
```

In general, you should add extra permissions only to the applications or modules that require them, not to all applications deployed to a domain. For example:

```
grant "file:${com.sun.aas.instanceRoot}/applications/j2ee-apps/MyApp/-" { ... }
```

For a module:

```
grant "file:${com.sun.aas.instanceRoot}/applications/j2ee-modules/MyModule/-" {
... }
```

Do not add extra permissions to the default set (the grant block with no codebase, which applies to all code). Instead, add a new grant block with a codebase specific to the application requiring the extra permissions, and only add the minimally necessary permissions in that block.

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

As noted in the J2EE specification, an application should provide documentation of the additional permissions it needs. If an application requires extra permissions but does not document the set it needs, contact the application author for details.

As a last resort, you can iteratively determine the permission set an application needs by observing `AccessControlException` occurrences in the server log. If this is not sufficient, you can add the `-Djava.security.debug=fail` JVM option to the domain. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* or the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

You can use the J2SE standard `policytool` or any text editor to edit the `server.policy` file. For more information, see:

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

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

http://java.sun.com/j2se/1.4/docs/guide/security/permissions.html

The Javadoc for the `Permission` class is at:

http://java.sun.com/j2se/1.4/docs/api/java/security/Permission.html

Configuring Message Security

In *message security*, security information travels along with the web services message. WSS in the SOAP layer is the use of XML Encryption and XML Digital Signatures to secure SOAP messages. WSS profiles the use of various security tokens including X.509 certificates, SAML assertions, and username/password tokens to achieve this.

Message layer security differs from transport layer security (which is discussed in the *Security* chapter of the J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`)) in that message layer security can be used to decouple message protection from message transport so that messages remain protected after transmission, regardless of how many hops they travel on.

WSS is a security mechanism that is applied at the message-layer in order to secure web services. For the purposes of this document, when we discuss WSS, we are talking about security for web services as described by the Oasis Web Services Security (WSS) specification. Message security for the Application Server follows this specification, which can be viewed at the following URL:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

For more information about message security, see the following:

- The *J2EE 1.4 Tutorial* chapter titled *Security*, which can be viewed from:`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`.
- The *Sun Java System Application Server Platform Edition 8.2 Administration Guide* chapter titled *Configuring Message Security*.

The following web services security topics are discussed in this section:

- "Message Security Responsibilities" on page 57
- "Application-Specific Message Protection" on page 58
- "Understanding and Running the Example Application" on page 61

Message Security Responsibilities

Message security responsibilities are assigned to the following:

Application Developer

The application developer can implement message security, but is not responsible for doing so. Message security can be set up by the System Administrator so that all web services are secured, or set up by the Application Deployer when the Application Server provider configuration is insufficient.

The application developer is responsible for the following:

- Determining if an application-specific policy is necessary for an application. If so, ensure that policy is satisfied at application assembly, or communicate the requirement for application-specific message security to the Application Deployer, or take care of implementing the application-specific policy.
- Determining if message security is necessary at the Application Server level. If so, ensure that need is communicated to the System Administrator, or take care of implementing message security at the Application-Server level.

Application Deployer

The application deployer is responsible for:

- Securing the application if it has not been appropriately secured by upstream roles (the developer or assembler) and only if an application-specific policy is appropriate for the application.
- Implementing application-specific security by adding the message security binding to the web service endpoint.
- Modifying Sun-specific deployment descriptors to add message binding information.

These security tasks are discussed in "Application-Specific Message Protection" on page 58. An example application using message security is discussed in "Understanding and Running the Example Application" on page 61.

System Administrator

The system administrator is responsible for:

- Configuring message security providers on the Application Server.
- Managing user databases.

- Managing keystore and truststore files.
- Configuring a Java Cryptography Extension (JCE) provider if using Encryption and running a version of the Java SDK prior to version 1.5.0.
- Installing the samples server in order to work with the example message security applications.

A system administrator uses the Admin Console or the `asadmin` tool to manage server security settings and `keytool` to manage certificates. System administrator tasks are discussed in the *Configuring Message Security* chapter of the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Application-Specific Message Protection

When the Application Server provided configuration is insufficient for your security needs, and you want to override the default protection, you can apply *application-specific message security* to a web service.

Application-specific security is implemented by adding the message security binding to the web service endpoint, whether it is an EJB or servlet web service endpoint. Modify Sun-specific XML files to add the message binding information.

For more details on message security binding for EJB web services, servlet web services, and clients, see the XML file descriptions in Appendix A, "Deployment Descriptor Files."

- For `sun-ejb-jar.xml`, see "The sun-ejb-jar.xml File" on page 288.
- For `sun-web.xml`, see "The sun-web.xml File" on page 285.
- For `sun-application-client.xml`, see "The sun-application-client.xml file" on page 297.

This section contains the following topics:

Using a Signature to Enable Message Protection for All Methods

To enable message protection for all methods using digital signature, update the `message-security-binding` element for the EJB web service endpoint in the application's `sun-ejb-jar.xml` file. In this file, add `request-protection` and `response-protection` elements, which are analogous to the `request-policy` and `response-policy` elements discussed in the *Configuring Message Security* chapter of the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*. In order to apply the same protection mechanisms for all methods, leave the method-name element blank. "Configuring Message Protection For a Specific Method Based on Digital Signatures" on page 59 discusses listing specific methods or using wildcard characters.

This section uses the sample application discussed in "Understanding and Running the Example Application" on page 61 to apply application-level message security in order to show only the differences necessary for protecting web services using various mechanisms.

▼ To enable message protection for all methods using digital signature

1 **In a text editor, open the application's `sun-ejb-jar.xml` file.**

For the `xms` example, this file is located in the directory *install-dir* `/samples/webservices/security/ejb/apps/xms/xms-ejb/src/conf`.

2 **Modify the `sun-ejb-jar.xml` file by adding the `message-security-binding` element as shown:**

```
<sun-ejb-jar>
  <enterprise-beans>
    <unique-id>1</unique-id>
    <ejb>
      <ejb-name>HelloWorld</ejb-name>
      <jndi-name>HelloWorld</jndi-name>
      <webservice-endpoint>
        <port-component-name>HelloIF</port-component-name>
        <endpoint-address-uri>service/HelloWorld</endpoint-address-uri>
        <message-security-binding auth-layer="SOAP">
          <message-security>
            <request-protection auth-source="content" />
            <response-protection auth-source="content"/>
          </message-security>
        </message-security-binding>
      </webservice-endpoint>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

3 **Compile, deploy, and run the application as described in "To Run the Sample Application" on page 62.**

Configuring Message Protection For a Specific Method Based on Digital Signatures

To enable message protection for a specific method, or for a set of methods that can be identified using a wildcard value, follow these steps. As in the example discussed in "Using a Signature to Enable Message Protection for All Methods" on page 58, to enable message protection for a specific method, update the `message-security-binding` element for the EJB web service endpoint in the application's `sun-ejb-jar.xml` file. To this file, add `request-protection` and `response-protection` elements, which are analogous to the `request-policy` and `response-policy` elements discussed in the *Configuring Message Security*

chapter of the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*. The *Administration Guide* includes a table listing the set and order of security operations for different request and response policy configurations.

This section uses the sample application discussed in "Understanding and Running the Example Application" on page 61 to apply application-level message security in order to show only the differences necessary for protecting web services using various mechanisms.

▼ To enable message protection for a particular method or set of methods using digital signature

1 In a text editor, open the application's `sun-ejb-jar.xml` file.

For the `xms` example, this file is located in the directory *install-dir*`/samples/webservices/security/ejb/apps/xms/xms-ejb/src/conf`.

2 Modify the `sun-ejb-jar.xml` file by adding the `message-security-binding` element as shown:

```
<sun-ejb-jar>
  <enterprise-beans>
  <unique-id>1</unique-id>
    <ejb>
      <ejb-name>HelloWorld</ejb-name>
      <jndi-name>HelloWorld</jndi-name>
      <webservice-endpoint>
        <port-component-name>HelloIF</port-component-name>
        <endpoint-address-uri>service/HelloWorld</endpoint-address-uri>
        <message-security-binding auth-layer="SOAP">
          <message-security>
            <message>
              <java-method>
                <method-name>ejbCreate</method-name>
              </java-method>
            </message>
            <message>
              <java-method>
                <method-name>sayHello</method-name>
              </java-method>
            </message>
            <request-protection auth-source="content" />
            <response-protection auth-source="content"/>
          </message-security>
        </message-security-binding>
      </webservice-endpoint>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

3 **Compile, deploy, and run the application as described in "To Run the Sample Application" on page 62.**

Understanding and Running the Example Application

This section discusses the WSS sample application, `xms`, which is located in the directory *install-dir*`/samples/webservices/security/ejb/apps/xms/`. This directory and this sample application is installed on your system only if you have selected to install the samples server when you installed the Application Server. If you have not installed the samples, see "To Set Up the Sample Application" on page 61.

The objective of this sample application is to demonstrate how a web service can be secured with WSS. The web service in the `xms` example is a simple web service implemented using a J2EE EJB endpoint and a web service endpoint implemented using a servlet. In this example, a service endpoint interface is defined with one operation, `sayHello`, which takes a string then sends a response with `Hello` prefixed to the given string. You can view the WSDL file for the service endpoint interface at *install-dir*`/samples/webservices/security/ejb/apps/xms/xms-ejb/src/conf/HelloWorld.wsdl`.

In this application, the client lookups the service using the JNDI name `java:comp/env/service/HelloWorld` and gets the port information using a static stub to invoke the operation using a given name. For the name Duke, the client gets the response `Hello Duke!`

This example shows how to use message security for web services at the Application Server level and at the application level. The WSS message security mechanisms implement message-level authentication (for example, XML digital signature and encryption) of SOAP web services invocations using the X.509 and username/password profiles of the OASIS WS-Security standard, which can be viewed from the following URL:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

This section includes the following topics:

▼ To Set Up the Sample Application

Before You Begin The example application is located in the directory *install-dir*`/samples/webservices/security/ejb/apps/xms/`. For ease of reference throughout the rest of this section, this directory is referred to as simply *app-dir*`/xms/`.

In order to have access to this sample application, you must have installed the `samples` server during installation of the Application Server. To check to see if the samples are installed, browse

to the directory *install-dir*`/samples/webservices/security/ejb/apps/xms/`. If this directory exists, you do not need to follow the steps in the following section. If this directory does not exist, the `samples` server is not installed, and must be installed for access to the sample application discussed here.

1. **Start the installation for the Application Server.**
2. **Click Next on the Welcome page.**
3. **Click Yes on the Software License Agreement page. Click Next.**
4. **Click Next to accept the installation directory, or change it to match the location where the Application Server is currently installed.**
5. **Select Continue to install to the same directory.**

   You want to do this because you want the `samples/` directory to be a subdirectory of the Application Server directory, *install-dir*`/samples/`.
6. **Reenter the Admin User Name and Password. Click Next.**

   You are on the page where you select to install just the samples.
7. **Deselect everything except Create Samples Server. Click Next.**
8. **Click Install Now to install the samples.**
9. **Click Finish to complete the installation.**

▼ To Run the Sample Application

1. **Make sure that the Application Server is running.**

   Message security providers are set up when the `asant` targets are run, so you don't need to configure these on the Application Server prior to running this example.
2. **If you are not running HTTP on the default port of 8080, change the WSDL file for the example to reflect the change, and change the `common.properties` file to reflect the change as well.**

   The WSDL file for this example is located at *install-dir*`/samples/webservices/security/ejb/apps/xms/xms-ejb/src/conf/HelloWorld.wsdl`. The port number is in the following section:

```
<service name="HelloWorld">
  <port name="HelloIFPort" binding="tns:HelloIFBinding">
    <soap:address location="http://localhost:8080/service/HelloWorld"/>
  </port>
</service>
```

Verify that the properties in the *install-dir*`/samples/common.properties` file are set properly for your installation and environment. If you need more description of this file, refer to the *Configuration* section for the web services security applications at *install-dir*`/samples/webservices/security/docs/common.html#Logging`.

3 **Change to the** *install-dir*`/samples/webservices/security/ejb/apps/xms/` **directory.**

4 **Run the following** `asant` **targets to compile, deploy, and run the example application:**

a. **To compile samples:**

```
asant
```

b. **To deploy samples:**

```
asant deploy
```

c. **To run samples:**

```
asant run
```

If the sample has compiled and deployed properly, you see the following response on your screen after the application has run:

```
run:[echo] Running the xms program:[exec] Established message level security :
Hello Duke!
```

5 **To undeploy the sample, run the following** `asant` **target:**

```
asant undeploy
```

All of the web services security examples use the same web service name (`HelloWorld`) and web service ports in order to show only the differences necessary for protecting web services using various mechanisms. Make sure to undeploy an application when you have completed running it, or you receive an `Already in Use` error and deployment failures when you try to deploy another web services example application.

Programmatic Login

Programmatic login allows a deployed J2EE application to invoke a login method. If the login is successful, a `SecurityContext` is established as if the client had authenticated using any of the conventional J2EE mechanisms.

Programmatic login is useful for an application having special needs that cannot be accommodated by any of the J2EE standard authentication mechanisms.

Note – Programmatic login is specific to the Application Server and not portable to other application servers.

This section contains the following topics:

Precautions

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

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

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

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

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

Granting Programmatic Login Permission

The `ProgrammaticLoginPermission` permission is required to invoke the programmatic login mechanism for an application. This permission is not granted by default to deployed applications because this is not a standard J2EE mechanism.

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

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

The *jar-file-path* is the path to the application's JAR file.

For more information about the `server.policy` file, see "The server.policy File" on page 54

The ProgrammaticLogin Class

The `com.sun.appserv.security.ProgrammaticLogin` class enables a user to perform login programmatically. This class has four `login` methods, two for servlets or JSP files and two for EJB components.

The login methods for servlets or JSP files have the following signatures:

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

public java.lang.Boolean login(String user, String password,
    String realm, javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response, boolean errors)
    throws java.lang.Exception
```

The login methods for EJB components have the following signatures:

```
public java.lang.Boolean login(String user, String password)

public java.lang.Boolean login(String user, String password,
    String realm, boolean errors) throws java.lang.Exception
```

All of these `login` methods:

- Perform the authentication
- Return `true` if login succeeded, `false` if login failed

The login occurs on the `realm` specified unless it is null, in which case the domain's default realm is used. The methods with no `realm` parameter use the domain's default realm.

If the `errors` flag is set to `true`, any exceptions encountered during the login are propagated to the caller. If set to `false`, exceptions are thrown.

On the client side, `realm` and `errors` parameters are ignored and the actual login does not occur until a resource requiring a login is accessed. A `java.rmi.AccessException` with `COBRA NO_PERMISSION` occurs if the actual login fails.

The logout methods for servlets or JSP files have the following signatures:

```
public java.lang.Boolean logout(HttpServletRequest request,
    HttpServletResponse response)

public java.lang.Boolean logout(HttpServletRequest request,
    HttpServletResponse response, boolean errors)
    throws java.lang.Exception
```

The logout methods for EJB components have the following signatures:

```
public java.lang.Boolean logout()

public java.lang.Boolean logout(boolean errors)
    throws java.lang.Exception
```

All of these `logout` methods return `true` if logout succeeded, `false` if logout failed.

If the `errors` flag is set to `true`, any exceptions encountered during the logout are propagated to the caller. If set to `false`, exceptions are thrown.

User Authentication for Single Sign-on

The single sign-on feature of the Application Server allows multiple web applications deployed to the same virtual server to share the user authentication state. With single sign-on enabled, users who log in to one web application become implicitly logged into other web applications on the same virtual server that require the same authentication information. Otherwise, users would have to log in separately to each web application whose protected resources they tried to access.

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

Single sign-on operates according to the following rules:

- Single sign-on applies to web applications configured for the same realm and virtual server. The realm is defined by the `realm-name` element in the `web.xml` file. For information about virtual servers, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- As long as users access only unprotected resources in any of the web applications on a virtual server, they are not challenged to authenticate themselves.
- As soon as a user accesses a protected resource in any web application associated with a virtual server, the user is challenged to authenticate himself or herself, using the login method defined for the web application currently being accessed.

- Once authenticated, the roles associated with this user are used for access control decisions across all associated web applications, without challenging the user to authenticate to each application individually.
- When the user logs out of one web application (for example, by invalidating the corresponding session), the user's sessions in all web applications are invalidated. Any subsequent attempt to access a protected resource in any application requires the user to authenticate again.

The single sign-on feature utilizes HTTP cookies to transmit a token that associates each request with the saved user identity, so it can only be used in client environments that support cookies.

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

- `sso-enabled` - If `false`, single sign-on is disabled for this virtual server, and users must authenticate separately to every application on the virtual server. The default is `true`.
- `sso-max-inactive-seconds` - Specifies the time after which a user's single sign-on record becomes eligible for purging if no client activity is received. Since single sign-on applies across several applications on the same virtual server, access to any of the applications keeps the single sign-on record active. The default value is 5 minutes (`300` seconds). Higher values provide longer single sign-on persistence for the users at the expense of more memory use on the server.
- `sso-reap-interval-seconds` - Specifies the interval between purges of expired single sign-on records. The default value is `60`.

Here is an example configuration with all default values:

```
<virtual-server id="server" ... >
     ...
    <property name="sso-enabled" value="true"/>
     <property name="sso-max-inactive-seconds" value="450"/>
     <property name="sso-reap-interval-seconds" value="80"/>
 </virtual-server>
```

Defining Roles

You define roles in the J2EE deployment descriptor file, `web.xml`, and the corresponding role mappings in the Application Server deployment descriptor file, `sun-application.xml` (or `sun-web.xml` for individually deployed web modules).

For more information regarding `web.xml` elements, see Chapter 13, "Deployment Descriptor," of the Java Servlet Specification, v2.4. For more information regarding `sun-web.xml` and `sun-application.xml` elements, see Appendix A, "Deployment Descriptor Files."

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

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

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

Note that for J2EE applications (EAR files), all security role mappings for the application modules must be specified in the `sun-application.xml` file. For individually deployed web modules, the roles are always specified in the `sun-web.xml` file. A role can be mapped to either specific principals or to groups (or both). The principal or group names used must be valid principals or groups in the current default realm.

◆ ◆ ◆ CHAPTER 3

Assembling and Deploying Applications

This chapter describes Sun Java System Application Server modules and how these modules are assembled separately or together in an application. This chapter also describes classloaders and tools for assembly and deployment.

The Application Server modules and applications include J2EE standard features and Application Server specific features. Only Application Server specific features are described in detail in this chapter.

The following topics are presented in this chapter:

Overview of Assembly and Deployment

Application assembly (also known as packaging) is the process of combining discrete components of an application into a single unit that can be deployed to a J2EE-compliant application server. A package can be classified either as a module or as a full-fledged application. This section covers the following topics:

Modules

A J2EE module is a collection of one or more J2EE components of the same container type (for example, web or EJB) with deployment descriptors of that type. One descriptor is J2EE standard, the other is Application Server specific. Types of J2EE modules are as follows:

- **Web Application Archive (WAR):** A web application is a collection of servlets, HTML pages, classes, and other resources that can be bundled and deployed to several J2EE application servers. A WAR file can consist of the following items: servlets, JSP files, JSP tag libraries, utility classes, static pages, client-side applets, beans, bean classes, and deployment descriptors (`web.xml` and optionally `sun-web.xml`).
- **EJB JAR File:** The EJB JAR file is the standard format for assembling enterprise beans. This file contains the bean classes (home, remote, local, and implementation), all of the utility classes, and the deployment descriptors (`ejb-jar.xml` and `sun-ejb-jar.xml`). If the EJB component is an entity bean with container managed persistence, a `.dbschema` file and a CMP mapping descriptor, `sun-cmp-mapping.xml`, must be included as well.
- **Application Client Container JAR File:** An ACC client is an Application Server specific type of J2EE client. An ACC client supports the standard J2EE Application Client specifications, and in addition, supports direct access to the Application Server. Its deployment descriptors are `application-client.xml` and `sun-application-client.xml`.
- **Resource RAR File:** RAR files apply to J2EE CA connectors. A connector module is like a device driver. It is a portable way of allowing EJB components to access a foreign enterprise system. Each Application Server connector has a J2EE XML file, `ra.xml`.

Package definitions must be used in the source code of all modules so the class loader can properly locate the classes after the modules have been deployed.

Because the information in a deployment descriptor is declarative, it can be changed without requiring modifications to source code. At run time, the J2EE server reads this information and acts accordingly.

The Application Server also supports lifecycle modules. See Chapter 10, “Developing Lifecycle Listeners,” for more information.

EJB JAR and Web modules can also be deployed separately, outside of any application, as in the following figure. EJB components are assembled in a JAR file with ejb-jar.xml and sun-ejb-jar.xml deployment descriptors. Web components are assembled in a WAR file with web.xml and sun-web.xml deployment descriptors. Both module types are deployed to the Application Server.

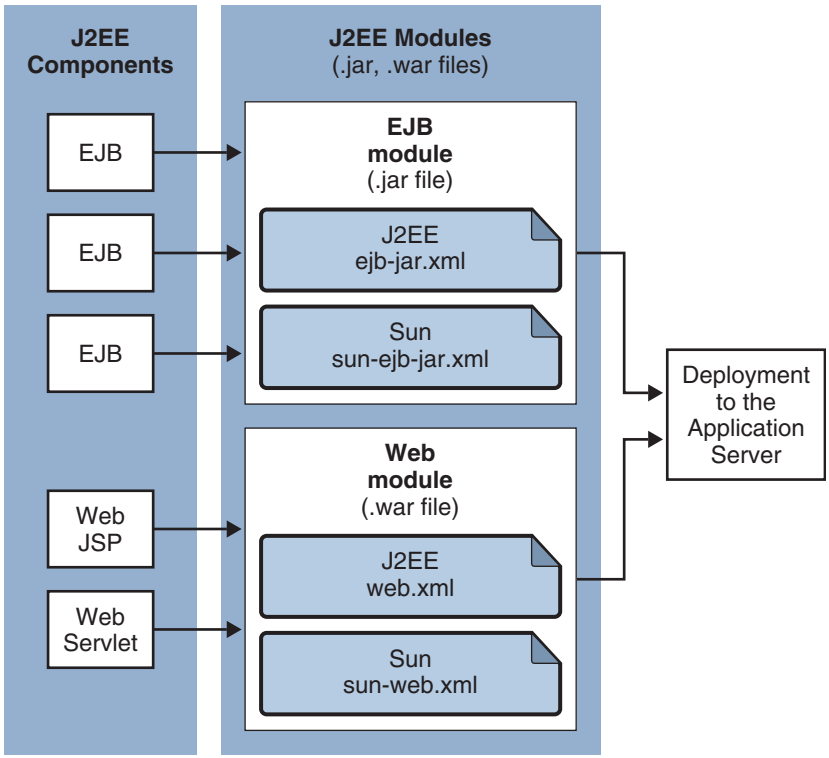

FIGURE 3–1 Module assembly and deployment

Applications

A J2EE application is a logical collection of one or more J2EE modules tied together by application deployment descriptors. Components can be assembled at either the module or the application level. Components can also be deployed at either the module or the application level.

The following diagram illustrates how components are assembled into modules and then assembled into an Application Server application and deployed. EJB components are assembled in a JAR file with ejb-jar.xml and sun-ejb-jar.xml deployment descriptors. Web components are assembled in a WAR file with web.xml and sun-web.xml deployment descriptors. An application client is assembled in a JAR file with application-client.xml and sun-application-client.xml deployment descriptors. A resource adapter is assembled in a RAR file with a ra.xml deployment descriptor. All modules are assembled in an EAR file and deployed to the Application Server.

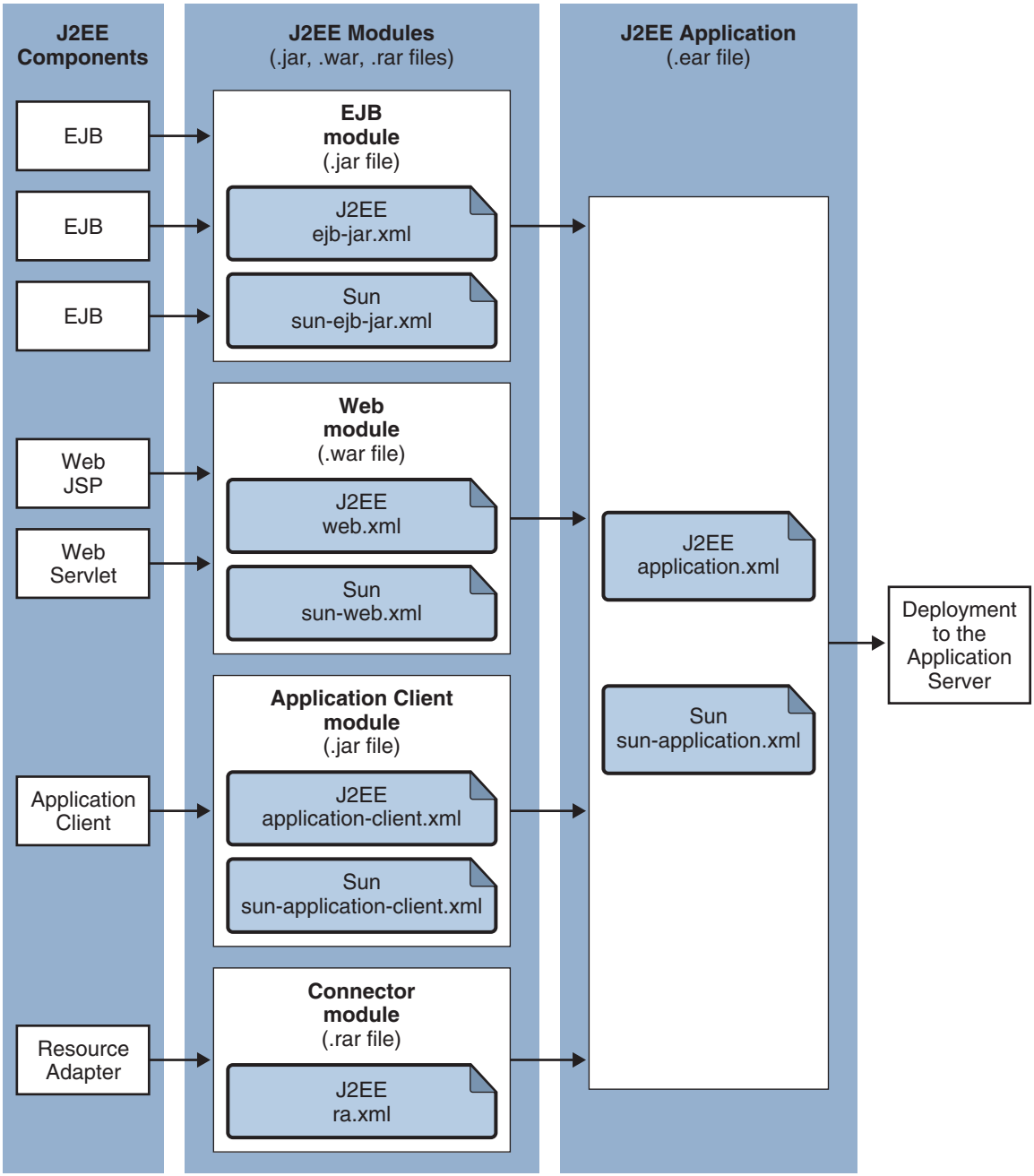

FIGURE 3–2 Application assembly and deployment

Each module has an Application Server deployment descriptor and a J2EE deployment descriptor. The Application Server uses the deployment descriptors to deploy the application components and to register the resources with the Application Server.

An application consists of one or more modules, an optional Application Server deployment descriptor, and a required J2EE application deployment descriptor. All items are assembled, using the Java ARchive (`.jar`) file format, into one file with an extension of `.ear`.

J2EE Standard Descriptors

The J2EE platform provides assembly and deployment facilities. These facilities use WAR, JAR, and EAR files as standard packages for components and applications, and XML-based deployment descriptors for customizing parameters.

J2EE standard deployment descriptors are described in the J2EE specification, v1.4. You can find the specification at `http://java.sun.com/products/`.

To check the correctness of these deployment descriptors prior to deployment, see "The Deployment Descriptor Verifier" on page 84.

The following table shows where to find more information about J2EE standard deployment descriptors.

TABLE 3–1 J2EE Standard Descriptors

Deployment Descriptor	Where to Find More Information
`application.xml`	Java 2 Platform Enterprise Edition Specification, v1.4, Chapter 8, "Application Assembly and Deployment - J2EE:application XML DTD"
`web.xml`	Java Servlet Specification, v2.4 Chapter 13, "Deployment Descriptor," and JavaServer Pages Specification, v2.0, Chapter 7, "JSP Pages as XML Documents," and Chapter 5, "Tag Extensions"
`ejb-jar.xml`	Enterprise JavaBeans Specification, v2.1, Chapter 16, "Deployment Descriptor"
`application-client.xml`	Java 2 Platform Enterprise Edition Specification, v1.4, Chapter 9, "Application Clients - J2EE:application-client XML DTD"
`ra.xml`	Java 2 Enterprise Edition, J2EE Connector Architecture Specification, v1.0, Chapter 10, "Packaging and Deployment."

Sun Java System Application Server Descriptors

The Application Server uses additional deployment descriptors for configuring features specific to the Application Server. The `sun-application.xml`, `sun-web.xml`, and `sun-cmp-mappings.xml` files are optional; all the others are required.

To check the correctness of these deployment descriptors prior to deployment, see "The Deployment Descriptor Verifier" on page 84.

The following table lists the Application Server deployment descriptors and their DTD files. For complete descriptions of these files, see Appendix A, "Deployment Descriptor Files."

TABLE 3–2 Sun Java System Application Server Descriptors

Deployment Descriptor	DTD File	Description
`sun-application.xml`	`sun-application_1_4-0.dtd`	Configures an entire J2EE application (EAR file).
`sun-web.xml`	`sun-web-app_2_4-1.dtd`	Configures a web application (WAR file).
`sun-ejb-jar.xml`	`sun-ejb-jar_2_1-1.dtd`	Configures an enterprise bean (EJB JAR file).
`sun-cmp-mappings.xml`	`sun-cmp-mapping_1_2.dtd`	Configures container-managed persistence for an enterprise bean.
`sun-application-client.xml`	`sun-application-client_1_4-1.dtd`	Configures an Application Client Container (ACC) client (JAR file).
`sun-acc.xml`	`sun-application-client-container_1_0.dtd`	Configures the Application Client Container.

Naming Standards

Names of applications and individually deployed EJB JAR, WAR, and connector RAR modules must be unique within an Application Server domain. Modules of the same type within an application must have unique names. In addition, for entity beans that use CMP, `.dbschema` file names must be unique within an application.

If you do not explicitly specify a name, the default name is the first portion of the file name (without the `.war` or `.jar` extension). Modules of different types can have the same name within an application, because the directories holding the individual modules are named with `_jar`, `_war` and `_rar` suffixes. This is the case when you use the Administration Console, the `asadmin` command, or the deploytool to deploy an application or module. See "Tools for Deployment" on page 93.

Make sure your package and file names do not contain spaces or characters that are illegal for your operating system.

If you are writing your own JSR 88 client to deploy applications to the Application Server using the following API, the name of the application is taken from the `display-name` entry in the

J2EE standard deployment descriptor, because there is no file name in this case. If the `display-name` entry is not present, the Application Server creates a temporary file name and uses that name to deploy the application.

```
javax.enterprise.deploy.spi.DeploymentManager.distribute(Target[],
InputStream, InputStream)
```

Neither the Administration Console, the `asadmin` command, nor the deploytool uses this API.

For more information about JSR 88, see the JSR 88 page at `http://jcp.org/en/jsr/detail?id=88`.

Directory Structure

When you deploy an application, the application is expanded to an open directory structure, and the directories holding the individual modules are named with `_jar`, `_war` and `_rar` suffixes. If you use the `asadmin deploydir` command to deploy a directory instead of an EAR file, your directory structure must follow this same convention.

Module and application directory structures follow the structure outlined in the J2EE specification. Here is an example directory structure of a simple application containing a web module, an EJB module, and a client module.

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

```
   |--+ META-INF/
      |--- MANIFEST.MF
      |--- application-client.xml
      |--- sun-application-client.xml
```

Here is an example directory structure of an individually deployed connector module.

```
+ MyConnector/
|--- readme.html
|--- ra.jar
|--- client.jar
|--- win.dll
|--- solaris.so
|--+ META-INF/
   |--- MANIFEST.MF
   |--- ra.xml
```

Runtime Environments

Whether you deploy an individual module or an application, deployment affects both the file system and the server configuration.

Module Runtime Environment

The following figure illustrates the environment for individually deployed module-based deployment.

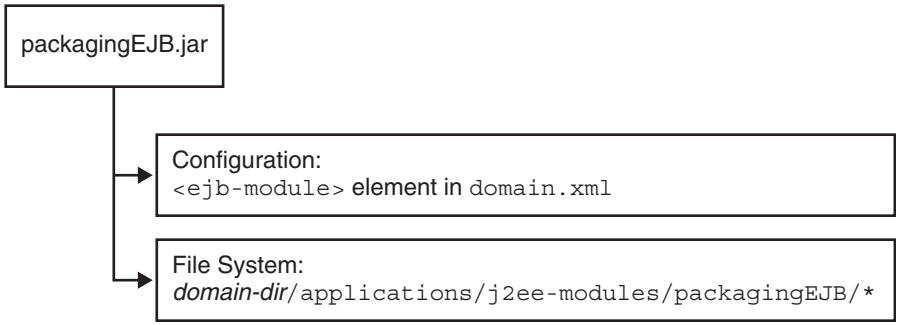

FIGURE 3–3 Module runtime environment

For file system entries, modules are extracted as follows:

domain-dir`/applications/j2ee-modules/`*module-name*
domain-dir`/generated/ejb/j2ee-modules/`*module-name*
domain-dir`/generated/jsp/j2ee-modules/`*module-name*

The `applications` directory contains the directory structures described in "Directory Structure" on page 75. The `generated/ejb` directory contains the stubs and ties that an ACC client needs to access the module; the `generated/jsp` directory contains compiled JSP files.

Lifecycle modules (see Chapter 10, "Developing Lifecycle Listeners") are extracted as follows:

domain-dir`/applications/lifecycle-modules/`*module-name*

Configuration entries are added in the `domain.xml` file as follows:

```
<server>
   <applications>
      <type-module>
         ...module configuration...
      </type-module>
   </applications>
</server>
```

The *type* of the module in `domain.xml` can be `lifecycle`, `ejb`, `web`, or `connector`. For details about `domain.xml`, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Application Runtime Environment

The following figure illustrates the environment for application-based deployment.

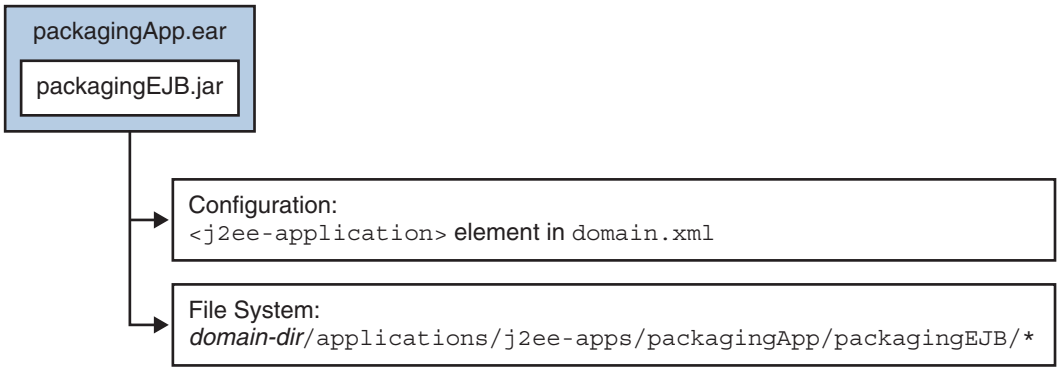

FIGURE 3–4 Application runtime environment

For file system entries, applications are extracted as follows:

domain-dir`/applications/j2ee-apps/`*app-name*
domain-dir`/generated/ejb/j2ee-apps/`*app-name*
domain-dir`/generated/jsp/j2ee-apps/`*app-name*

The `applications` directory contains the directory structures described in "Directory Structure" on page 75. The `generated/ejb` directory contains the stubs and ties that an ACC client needs to access the module; the `generated/jsp` directory contains compiled JSP files.

Configuration entries are added in the `domain.xml` file as follows:

```
<server>
   <applications>
      <j2ee-application>
         ...application configuration...
      </j2ee-application>
   </applications>
</server>
```

For details about `domain.xml`, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Classloaders

Understanding Application Server classloaders can help you determine where and how you can position supporting JAR and resource files for your modules and applications.

In a Java Virtual Machine (JVM), the classloaders dynamically load a specific Java class file needed for resolving a dependency. For example, when an instance of `java.util.Enumeration` needs to be created, one of the classloaders loads the relevant class into the environment. This section includes the following topics:

The Classloader Hierarchy

Classloaders in the Application Server runtime follow a hierarchy that is illustrated in the following figure and fully described in Table 3–3.

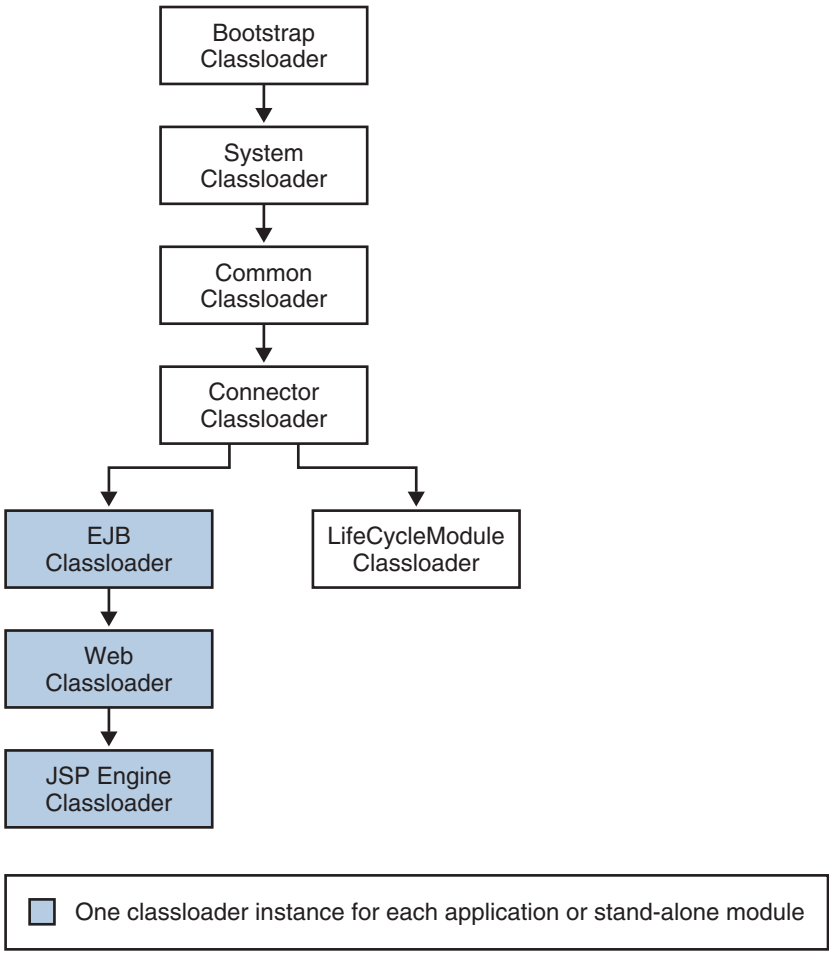

FIGURE 3–5 Classloader runtime hierarchy

TABLE 3–3 Sun Java System Application Server Classloaders

Classloader	Description
Bootstrap	The Bootstrap Classloader loads all the JDK classes. It is parent to the System Classloader.
System	The System Classloader loads most of the core Application Server classes. It is parent to the Common Classloader. It is created based on the `classpath-prefix`, `server-classpath`, and `classpath-suffix` attributes of the `java-config` element in the `domain.xml` file. The environment classpath is included if `env-classpath-ignored="false"` is set in the `java-config` element.
Common	The Common Classloader loads into the system classpath classes in the *domain-dir*`/lib/classes` directory, followed by JAR and ZIP files in the *domain-dir*`/lib` directory. It is parent to the Connector Classloader. No special classpath settings are required. The existence of these directories is optional; if they don't exist, the Common Classloader is not created.
Connector	The Connector Classloader is a single class loader instance that loads individually deployed connector modules, which are shared across all applications. It is parent to the LifeCycleModule Classloader and the EJB Classloader.
LifeCycleModule	The LifeCycleModule Classloader is the parent class loader for lifecycle modules. Each lifecycle module's classpath is used to construct its own class loader.
EJB	The EJB Classloader loads the enabled EJB classes in a specific enabled EJB module or J2EE application. One instance of this class loader is present in each class loader universe. The EJB Classloader is created with a list of URLs that point to the locations of the classes it needs to load. It is parent to the Web Classloader.
Web	The Web Classloader loads the servlets and other classes in a specific enabled web module or J2EE application. One instance of this class loader is present in each class loader universe. The Web Classloader is created with a list of URLs that point to the locations of the classes it needs to load. It is parent to the JSP Engine Classloader.
JSP Engine	The JSP Engine Classloader loads compiled JSP classes of enabled JSP files. One instance of this class loader is present in each class loader universe. The JSP Engine Classloader is created with a list of URLs that point to the locations of the classes it needs to load.

Note that this is not a Java inheritance hierarchy, but a delegation hierarchy. In the delegation design, a class loader delegates classloading to its parent before attempting to load a class itself. A class loader parent can be either the System Classloader or another custom class loader. If the parent class loader can't load a class, the `findClass()` method is called on the class loader subclass. In effect, a class loader is responsible for loading only the classes not available to the parent.

The Servlet specification recommends that the Web Classloader look in the local class loader before delegating to its parent. You can make the Web Classloader follow the delegation model

in the Servlet specification by setting `delegate="false"` in the "class-loader" on page 311 element of the `sun-web.xml` file. It's safe to do this only for a web module that does not interact with any other modules.

The default value is `delegate="true"`, which causes the Web Classloader to delegate in the same manner as the other classloaders. You must use `delegate="true"` for a web application that accesses EJB components or that acts as a web service client or endpoint. For details about `sun-web.xml`, see "The sun-web.xml File" on page 285.

Classloader Universes

Access to components within applications and modules installed on the server occurs within the context of isolated class loader universes, each of which has its own EJB, Web, and JSP Engine classloaders.

- **Application Universe:** Each J2EE application has its own class loader universe, which loads the classes in all the modules in the application.
- **Individually Deployed Module Universe:** Each individually deployed EJB JAR, web WAR, or lifecycle module has its own class loader universe, which loads the classes in the module.

Note – A resource such as a file that is accessed by a servlet, JSP, or EJB component must be in a directory pointed to by the class loader's classpath. For example, the web class loader's classpath includes these directories:

module-name`/WEB-INF/classes`
module-name`/WEB-INF/lib`

If a servlet accesses a resource, it must be in one of these directories or it is not loaded.

Note – In iPlanet Application Server 6.*x*, individually deployed modules shared the same class loader. In subsequent Application Server versions, each individually deployed module has its own class loader universe.

Circumventing Classloader Isolation

Since each application or individually deployed module class loader universe is isolated, an application or module cannot load classes from another application or module. This prevents two similarly named classes in different applications from interfering with each other.

To circumvent this limitation for libraries, utility classes, or individually deployed modules accessed by more than one application, you can include the relevant path to the required classes in one of these ways:

- "Using the System Classloader" on page 82

Using the System Classloader

To use the System Classloader, do one of the following, then restart the server:

- Use the Administration Console. Select the JVM Settings component under the relevant configuration, select the Path Settings tab, and edit the Classpath Suffix field. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Edit the `classpath-suffix` attribute of the `java-config` element in the `domain.xml` file. For details about `domain.xml`, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Using the System Classloader makes an application or module accessible to any other application or module across the domain.

Using the Common Classloader

To use the Common Classloader, copy the JAR and ZIP files into the *domain-dir*`/lib` directory or copy the `.class` files into the *domain-dir*`/lib/classes` directory, then restart the server.

Using the Common Classloader makes an application or module accessible to any other application or module across the domain.

Using the Java Optional Package Mechanism

To use the Java optional package mechanism, copy the JAR and ZIP files into the *domain-dir*`/lib/ext` directory, then restart the server.

Using the Java optional package mechanism makes an application or module accessible to any other application or module across the domain.

For example, this is the recommended way of adding JDBC drivers to the Application Server. For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

Packaging the Client JAR for One Application in Another Application

By packaging the client JAR for one application in a second application, you allow an EJB or web component in the second application to call an EJB component in the first (dependent) application, without making either of them accessible to any other application or module.

As an alternative for a production environment, you can have the Common Classloader load client JAR of the dependent application as described in "Using the Common Classloader" on page 82 restart the server to make the dependent application accessible, and it is accessible across the domain.

▼ To package the client JAR for one application in another application

1. **Deploy the dependent application.**

2. **Add the dependent application's client JAR file to the calling application.**

   - For a calling EJB component, add the client JAR file at the same level as the EJB component. Then add a `Class-Path` entry to the `MANIFEST.MF` file of the calling EJB component. The `Class-Path` entry has this syntax:

     ```
     Class-Path: filepath1.jar filepath2.jar ...
     ```

     Each *filepath* is relative to the directory or JAR file containing the `MANIFEST.MF` file. For details, see the J2EE specification, section 8.1.1.2, "Dependencies."
   - For a calling web component, add the client JAR file under the `WEB-INF/lib` directory.

3. **If you need to package the client JAR with both the EJB and web components, set `delegate="true"` in the `class-loader` element of the `sun-web.xml` file.**

   This changes the Web Classloader so it follows the standard class loader delegation model and delegates to its parent before attempting to load a class itself.

   For most applications, packaging the client JAR file with the calling EJB component is sufficient. You do not need to package the client JAR file with both the EJB and web components unless the web component is directly calling the EJB component in the dependent application.

4. **Deploy the calling application.**

   The calling EJB or web component must specify in its `sun-ejb-jar.xml` or `sun-web.xml` file the JNDI name of the EJB component in the dependent application. Using an `ejb-link` mapping does not work when the EJB component being called resides in another application.

Assembling Modules and Applications

Assembling (or packaging) modules and applications in Application Server conforms to all of the customary J2EE-defined specifications. The only difference is that when you assemble in Application Server, you include Application Server specific deployment descriptors that enhance the functionality of the Application Server.

For example, when you assemble an EJB JAR module, you must create two deployment descriptor files with these names: `ejb-jar.xml` and `sun-ejb-jar.xml` (both required). If the

EJB component is an entity bean with container-managed persistence, you can also create a `.dbschema` file and a `sun-cmp-mapping.xml` file, but these are not required. For more information about `sun-ejb-jar.xml` and `sun-cmp-mapping.xml`, see Appendix A, "Deployment Descriptor Files."

Note – According to the J2EE specification, section 8.1.1.2, "Dependencies," you cannot package utility classes within an individually deployed EJB module. Instead, package the EJB module and utility JAR within an application using the JAR Extension Mechanism Architecture. For other alternatives, see "Circumventing Classloader Isolation" on page 81.

The Application Server provides these tools for assembling and verifying a module or an application:

deploytool

You can use the deploytool, provided with the Application Server, to assemble J2EE applications and modules, configure deployment parameters, perform simple static checks, and deploy the final result. For more information about using the deploytool, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`.

Apache Ant

Ant can help you assemble and deploy modules and applications. For details, see "asant Assembly and Deployment Tool" on page 98.

NetBeans IDE

You can use the NetBeans IDE to assemble J2EE applications and modules. For more information about using the NetBeans IDE, see `http://www.netbeans.org`.

The Deployment Descriptor Verifier

The verifier tool validates both J2EE and Application Server specific deployment descriptors against their corresponding DTD files and gives errors and warnings if a module or application is not J2EE and Application Server compliant. You can verify deployment descriptors in EAR, WAR, RAR, and JAR files.

The verifier tool is not simply an XML syntax verifier. Rules and interdependencies between various elements in the deployment descriptors are verified. Where needed, user application classes are introspected to apply validation rules.

The verifier is integrated into Application Server deployment, the deploytool, and the "sun-appserv-deploy" on page 99 Ant task. You can also run it as a stand-alone utility from the command line. The verifier is located in the *install-dir*`/bin` directory.

When you run the verifier during Application Server deployment, the output of the verifier is written to the *tempdir*`/verifier-results/` directory, where *tempdir* is the temporary directory defined in the operating system. Deployment fails if any failures are reported during the verification process. The verifier also logs information about verification progress to the standard output.

For details on all the assertions tested by the verifier, see the assertions documentation provided at `http://java.sun.com/j2ee/avk/index.html`.

Tip – Using the verifier tool can help you avoid runtime errors that are difficult to debug.

This section covers the following topics:

- "Command Line Syntax" on page 85
- "Ant Integration" on page 86
- "Sample Results Files" on page 87

Command Line Syntax

The verifier tool's syntax is as follows:

`verifier [`*options*`]` *file*

The *file* can be an EAR, WAR, RAR, or JAR file.

The following table shows the *options* for the `verifier` tool.

TABLE 3–4 Verifier Options

Short Form	Long Form	Description
`-v`	`--verbose`	Turns on verbose mode.
`-d` *output-dir*	`--destdir`	Writes test results to the *output-dir*, which must already exist. By default, the results files are created in the current directory.

TABLE 3–4 Verifier Options *(Continued)*

Short Form	Long Form	Description
`-r` *level*	`--reportlevel` *level*	Sets the output report *level* to one of the following values: ■ `a` or `all` - Reports all results. This is the default in both verbose and non verbose modes. ■ `w` or `warnings` - Reports only warning and failure results. ■ `f` or `failures` - Reports only failure results.
`-n`	`--notimestamp`	Does not append the timestamp to the output file name.
`-?`	`--help`	Displays help for the verifier command. If you use this option, you do not need to specify an EAR, WAR, RAR, or JAR file.
`-V`	`--version`	Displays the verifier tool version. If you use this option, you do not need to specify an EAR, WAR, RAR, or JAR file.
`-u`	`--gui`	Opens a graphical interface for performing verification. If you use this option, you do not need to specify an EAR, WAR, RAR, or JAR file. For more information, see the verifier online help.

For example, the following command runs the verifier in verbose mode and writes all the results of static verification of the `ejb.jar` file to the output directory `ResultsDir`:

```
verifier -v -r a -d ResultsDir ejb.jar
```

The results files are `ejb.jar_verifier.`*timestamp*`.txt` and `ejb.jar_verifier.`*timestamp*`.xml`. The format of the *timestamp* is yyyyMMddhhmmss.

If the verifier runs successfully, a result code of `0` is returned. This does not mean that no verification errors occurred. A nonzero error code is returned if the verifier fails to run.

Ant Integration

You can integrate the verifier into an Ant build file as a target and use the Ant call feature to call the target each time an application or module is assembled. This is because the `main` method in `com.sun.enterprise.tools.verifier.Verifier` is callable from user Ant scripts. The `main` method accepts the arguments described in Table 3–4.

Example code for an Ant verify target is as follows:

```
<target name="verify">
 <echo message="Verification Process for ${testfile}"/>
 <java classname="com.sun.enterprise.tools.verifier.Verifier"
         fork="yes">
     <sysproperty key="com.sun.enterprise.home"
         value="${appserv.home}"/>
     <sysproperty key="verifier.xsl"
```

```
        value="${appserv.home}/verifier/config" />
    <!-- uncomment the following for verbose output -->
    <!--<arg value="-v"/>-->
    <arg value="${assemble}/${ejbjar}" />
    <classpath path="${appserv.cpath}:${java.class.path}"/>
 </java>
</target>
```

Sample Results Files

Here is a sample results XML file:

```
<static-verification>
 <ejb>
     <failed>
         <test>
             <test-name>
tests.ejb.session.TransactionTypeNullForContainerTX
            </test-name>
             <test-assertion>
Session bean with bean managed transaction demarcation test
            </test-assertion>
             <test-description>
For [ TheGreeter ] Error: Session Beans [ TheGreeter ] with
[ Bean ] managed transaction demarcation should not have
container transactions defined.
             </test-description>
         </test>
     </failed>
 </ejb>
...
</static-verification>
```

Here is a sample results TXT file:

```
---------------------------
STATIC VERIFICATION RESULTS
---------------------------
---------------------------------
NUMBER OF FAILURES/WARNINGS/ERRORS
---------------------------------

# of Failures : 3
# of Warnings : 6
# of Errors : 0
-----------------------------
RESULTS FOR EJB-RELATED TESTS
-----------------------------
```

```
--------------
FAILED TESTS :
--------------

Test Name : tests.ejb.session.TransactionTypeNullForContainerTX
Test Assertion : Session bean with bean managed transaction demarcation test
Test Description : For [ TheGreeter ]
Error: Session Beans [ TheGreeter ] with [ Bean ] managed transaction
demarcation should not have container transactions defined.

...
---------------
PASSED TESTS :
---------------

Test Name : tests.ejb.session.ejbcreatemethod.EjbCreateMethodStatic
Test Assertion : Each session Bean must have at least one non-static
ejbCreate method test
Test Description : For [ TheGreeter ] For EJB Class
[ samples.helloworld.ejb.GreeterEJB ] method [ ejbCreate ]
[ samples.helloworld.ejb.GreeterEJB ] properly declares non-static
ejbCreate(...) method.

...
-----------
WARNINGS :
-----------

Test Name : tests.ejb.businessmethod.BusinessMethodException
Test Assertion : Enterprise bean business method throws RemoteException test
Test Description :

Test Name : tests.ejb.ias.beanpool.IASEjbBeanPool
Test Assertion :
Test Description : WARNING [IAS-EJB ejb] : bean-pool should be defined for
Stateless Session and Message Driven Beans

...
---------------------
NOTAPPLICABLE TESTS :
---------------------

Test Name : tests.ejb.entity.pkmultiplefield.PrimaryKeyClassFieldsCmp
Test Assertion : Ejb primary key class properly declares all class fields
within subset of the names of the container-managed fields test.
Test Description : For [ TheGreeter ] class com.sun.enterprise.tools.
verifier.tests.ejb.entity.pkmultiplefield.PrimaryKeyClassFieldsCmp
expected Entity bean, but called with Session.
```

```
Test Name : tests.ejb.entity.ejbcreatemethod.EjbCreateMethodReturn
Test Assertion : Each entity Bean can have zero or more ejbCreate
methods which return primary key type test
Test Description : For [ TheGreeter ] class com.sun.enterprise.tools.
verifier.tests.ejb.entity.ejbcreatemethod.EjbCreateMethodReturn
expected Entity bean, but called with Session bean.

...
---------------------------------
RESULTS FOR OTHER XML-RELATED TESTS
---------------------------------
---------------
PASSED TESTS :
---------------

Test Name : tests.dd.ParseDD
Test Assertion : Test parses the deployment descriptor using a SAX
parser to avoid the dependency on the DOL
Test Description : PASSED [EJB] : [ remote ] and [ home ] tags present.
PASSED [EJB]: session-type is Stateless.
PASSED [EJB]: trans-attribute is NotSupported.
PASSED [EJB]: transaction-type is Bean.

...
```

Deploying Modules and Applications

This section describes the different ways to deploy J2EE applications and modules to the Application Server. It covers the following topics:

Deployment Errors

If an error occurs during deployment, the application or module is not deployed. If a module within an application contains an error, the entire application is not deployed. This prevents a partial deployment that could leave the server in an inconsistent state.

The Deployment Life Cycle

After an application is initially deployed, it can be modified and reloaded, redeployed, disabled, re-enabled, and finally undeployed (removed from the server). This section covers the following topics related to the deployment life cycle:

Note – You can overwrite a previously deployed application by using the `--force` option of `asadmin deploy` or by checking the appropriate box in the Administration Console during deployment. However, you must remove a preconfigured resource before you can update it.

Dynamic Deployment

You can deploy, redeploy, and undeploy an application or module without restarting the server. This is called dynamic deployment. Although primarily for developers, dynamic deployment can be used in operational environments to bring new applications and modules online without requiring a server restart.

Whenever a redeployment is done, the sessions at that transit time become invalid. The client must restart the session.

Disabling a Deployed Application or Module

You can disable a deployed application or module without removing it from the server. Disabling an application makes it inaccessible to clients.

To disable an application or module using the `asadmin disable` command, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

▼ To disable an application or module in the Administration Console

1. **Open the Applications component.**
2. **Go to the page for the type of application or module.**
   For example, for a web application, go to the Web Applications page.
3. **Click on the name of the application or module you wish to disable.**
4. **Uncheck the Status Enabled box.**

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Dynamic Reloading

If dynamic reloading is enabled (it is by default), you do not have to redeploy an application or module when you change its code or deployment descriptors. All you have to do is copy the changed JSP or class files into the deployment directory for the application or module. The server checks for changes periodically and redeploys the application, automatically and dynamically, with the changes.

This is useful in a development environment, because it allows code changes to be tested quickly. In a production environment, however, dynamic reloading might degrade performance. In addition, whenever a reload is done, the sessions at that transit time become invalid. The client must restart the session.

▼ To enable dynamic reloading in the Administration Console

1. **Select the Application Settings component under the relevant configuration.**
2. **Check the Reload Enabled box to enable dynamic reloading.**
3. **Enter a number of seconds in the Reload Poll Interval field.**
   This sets the interval at which applications and modules are checked for code changes and dynamically reloaded. The default is 2.

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

▼ To reload code or deployment descriptor changes

1. **Create an empty file named `.reload` at the root of the deployed application or module.**

   For an application:

   domain-dir`/applications/j2ee-apps/`*app-name*`/.reload`

   For an individually deployed module:

   domain-dir`/applications/j2ee-modules/`*module-name*`/.reload`

2. **Explicitly update the `.reload` file's timestamp (`touch .reload` in UNIX) each time you make changes.**

Automatic Deployment

Automatic deployment, also called *autodeployment*, involves copying an application or module file (JAR, WAR, RAR, or EAR) into a special directory, where it is automatically deployed by the Application Server. To undeploy an automatically deployed application or module, simply remove its file from the special autodeployment directory. This is useful in a development environment, because it allows new code to be tested quickly.

Autodeployment is enabled by default.

▼ To enable and configure or to disable autodeployment

1. **Select the Application Settings component under the relevant configuration.**

2. **Check the Auto Deploy Enabled box to enable autodeployment, or uncheck this box to disable autodeployment.**

3. **Enter a number of seconds in the Auto Deploy Poll Interval field.**

   This sets the interval at which applications and modules are checked for code changes and dynamically reloaded. The default is 2.

4. **You can change the Auto Deploy Directory if you like.**

   You can enter an absolute or relative path. A relative path is relative to *domain-dir*. The default is *domain-dir*`/autodeploy`.

5. **You can check the Verifier Enabled box to verify your deployment descriptor files. This is optional.**

   For details about the verifier, see "The Deployment Descriptor Verifier" on page 84.

6. **Check the Precompile Enabled box to precompile any JSP files.**

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Tools for Deployment

This section discusses the various tools that can be used to deploy modules and applications. The deployment tools include:

- "Apache Ant" on page 93
- "The deploytool" on page 93
- "JSR 88" on page 93
- "The asadmin Command" on page 42
- "The Administration Console" on page 43

Apache Ant

Ant can help you assemble and deploy modules and applications. For details, see "asant Assembly and Deployment Tool" on page 98.

The deploytool

You can use the deploytool, provided with Application Server, to assemble J2EE applications and modules, configure deployment parameters, perform simple static checks, and deploy the final result. For more information about using the deploytool, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`.

JSR 88

You can write your own JSR 88 client to deploy applications to the Application Server. For more information, see the JSR 88 page at `http://jcp.org/en/jsr/detail?id=88`.

See "Naming Standards" on page 74 for application and module naming considerations.

The asadmin Command

You can use the `asadmin deploy` or `asadmin deploydir` command to deploy or undeploy applications and individually deployed modules on local servers. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

To deploy a lifecycle module, see "Deploying a Lifecycle Module" on page 95.

Note – On Windows, if you are deploying a directory on a mapped drive, you must be running the Application Server as the same user to which the mapped drive is assigned, or the Application Server won't see the directory.

The Administration Console

You can use the Administration Console to deploy modules and applications to both local and remote Application Server sites.

▼ To use the Administration Console for deployment

1. **Open the Applications component.**
2. **Go to the page for the type of application or module.**
   For example, for a web application, go to the Web Applications page.
3. **Click on the Deploy button.**
   You can also undeploy, enable, or disable an application or module from this page.

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

To deploy a lifecycle module, see "Deploying a Lifecycle Module" on page 95.

Deployment by Module or Application

You can deploy applications or individual modules that are independent of applications. The runtime and file system implications of application-based or individual module-based deployment are described in "Runtime Environments" on page 76.

Individual module-based deployment is preferable when components need to be accessed by:

- Other modules
- J2EE Applications
- ACC clients (Module-based deployment allows shared access to a bean from an ACC client, a servlet, or an EJB component.)

Modules can be combined into an EAR file and then deployed as a single module. This is similar to deploying the modules of the EAR independently.

Deploying a WAR Module

You deploy a WAR module as described in "Tools for Deployment" on page 93.

You can precompile JSP files during deployment by checking the appropriate box in the Administration Console or by using the `--precompilejsp` option of the `asadmin deploy` or `asadmin deploydir` command. The "sun-appserv-deploy" on page 99 and "sun-appserv-jspc" on page 111 Ant tasks also allow you to precompile JSP files.

You can keep the generated source for JSP files by adding the `-keepgenerated` flag to the `jsp-config` element in `sun-web.xml`. If you include this property when you deploy the WAR module, the generated source is kept in *domain-dir*`/generated/jsp/j2ee-apps/`*app-name*`/`*module-name* if it is in an application or *domain-dir*`/generated/jsp/j2ee-modules/`*module-name* if it is in an individually deployed web module.

For more information about JSP precompilation, see "Options for Compiling JSP Files" on page 142 "jsp-config" on page 342.

Deploying an EJB JAR Module

You deploy an EJB JAR module as described in "Tools for Deployment" on page 93.

You can keep the generated source for stubs and ties by adding the `-keepgenerated` flag to the `rmic-options` attribute of the `java-config` element in `domain.xml`. If you include this flag when you deploy the EJB JAR module, the generated source is kept in *domain-dir*`/generated/ejb/j2ee-apps/`*app-name*`/`*module-name* if it is in an application or *domain-dir*`/generated/ejb/j2ee-modules/`*module-name* if it is in an individually deployed EJB JAR module. For more information about the `-keepgenerated` flag, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Generation of stubs and ties is performed asynchronously, so unless you request their generation during deployment (for example, using the `--retrieve` option of the `asadmin deploy` command), stubs and ties are not guaranteed to be available immediately after deployment. You can use the `asadmin get-client-stubs` command to retrieve the stubs and ties whether or not you requested their generation during deployment. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Deploying a Lifecycle Module

For general information about lifecycle modules, see Chapter 10, "Developing Lifecycle Listeners."

You can deploy a lifecycle module using the following tools:

- In the Administration Console, open the Applications component and go to the Lifecycle Modules page. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-lifecycle-module` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Note – If the `is-failure-fatal` setting is set to `true` (the default is `false`), lifecycle module failure prevents server initialization or startup, but not shutdown or termination.

Deploying an Application Client

Deployment is only necessary for application clients that communicate with EJB components.

▼ To deploy an application client

1. **Assemble the necessary client files.**
2. **Assemble the EJB components to be accessed by the client.**
3. **Package the client and EJB components together in an application.**
4. **Deploy the application as described in "Tools for Deployment" on page 93.**
5. **Retrieve the client JAR file.**

   The client JAR file contains the ties and necessary classes for the ACC client.

   You can use the `--retrieve` option to get the client JAR file.

   You can also use the `asadmin get-client-stubs` command to retrieve the stubs and ties whether or not you requested their generation during deployment. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.
6. **Copy the client JAR file to the client machine, and set the APPCPATH environment variable on the client to point to this JAR.**

Next Steps To execute the client on the Application Server machine to test it, use the `appclient` script in the *install-dir*`/bin` directory.The only required option is `-client`. For example:

```
appclient -client converterClient.jar
```

The `-xml` parameter specifies the location of the `sun-acc.xml` file.

See Also For more detailed information about the `appclient` script, see Chapter 8, "Developing Java Clients."

▼ To prepare another machine for executing an application client

1. **You can use the `package-appclient` script in the *install-dir*`/bin` directory to create the ACC package JAR file. This is optional.**

   This JAR file is created in the *install-dir* `/lib/appclient` directory.

2. **Copy the ACC package JAR file to the client machine and unjar it.**

3. **Configure the `sun-acc.xml` file.**

   This file is located in the `appclient/appserv/lib/appclient` directory by default if you used the `package-appclient` script.

4. **Configure the `asenv.conf` (`asenv.bat` on Windows) file.**

   This file is located in `appclient/appserv/bin` by default if you used the `package-appclient` script.

5. **Copy the client JAR file to the client machine.**

   You are now ready to execute the client.

See Also For more detailed information about the `package-appclient` script, see Chapter 8, "Developing Java Clients."

Deploying a J2EE CA Resource Adapter

You deploy a connector module as described in "Tools for Deployment" on page 93. After deploying the module, you must configure it as described in Chapter 9, "Developing Connectors."

Access to Shared Frameworks

When J2EE applications and modules use shared framework classes (such as utility classes and libraries) the classes can be put in the path for the System Classloader or the Common Classloader rather than in an application or module. If you assemble a large, shared library into every module that uses it, the result is a huge file that takes too long to register with the server. In addition, several versions of the same class could exist in different classloaders, which is a waste of resources. For more information, see "Circumventing Classloader Isolation" on page 81.

asant Assembly and Deployment Tool

Apache Ant 1.6.5 is provided with Application Server and can be launched from the `bin` directory using the command `asant`. The Application Server also provides server-specific tasks for deployment, which are described in this section.

Make sure you have done these things before using `asant`:

- Include *install-dir*`/bin` in the PATH environment variable (`/usr/sfw/bin` for Sun Java Enterprise System on Solaris). The Ant script provided with the Application Server, `asant`, is located in this directory. For details on how to use `asant`, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual* and the sample applications documentation in the *install-dir*`/samples/docs/ant.html` file.
- If you are executing platform-specific applications, such as the `exec` or `cvs` task, the ANT_HOME environment variable must be set to the Ant installation directory.
  - The ANT_HOME environment variable for Sun Java Enterprise System must include the following:
    - `/usr/sfw/bin` - the Ant binaries (shell scripts)
    - `/usr/sfw/doc/ant` - HTML documentation
    - `/usr/sfw/lib/ant` - Java classes that implement Ant
  - The ANT_HOME environment variable for all other platforms is *install-dir*`/lib`.
- Set up your password file. The argument for the `passworfile` option of each Ant task is a file. This file contains the password attribute name and its value, in the following format:

  ```
  AS_ADMIN_PASSWORD=password
  ```

  For more information about password files, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

This section covers the following `asant`-related topics:

- "asant Tasks for Sun Java System Application Server" on page 98
- "Reusable Subelements" on page 114

For more information about Ant, see the Apache Software Foundation web site at `http://ant.apache.org/`.

For information about standard Ant tasks, see the Ant documentation at `http://ant.apache.org/manual/`.

asant Tasks for Sun Java System Application Server

Use the `asant` tasks provided by the Application Server for assembling, deploying, and undeploying modules and applications, and for configuring the server. The tasks are as follows:

sun-appserv-deploy

Deploys any of the following.

- Enterprise application (EAR file)
- Web application (WAR file)
- Enterprise Java Bean (EJB-JAR file)
- Enterprise connector (RAR file)
- Application client

Subelements

The following table describes subelements for the `sun-appserv-deploy` task. These are objects upon which this task acts.

TABLE 3–5 `sun-appserv-deploy` Subelements

Element	Description
"component" on page 114	A component to be deployed.
"fileset" on page 116	A set of component files that match specified parameters.

Attributes

The following table describes attributes for the `sun-appserv-deploy` task.

TABLE 3–6 `sun-appserv-deploy` Attributes

Attribute	Default	Description
`file`	none	(optional if a `component` or `fileset` subelement is present, otherwise required) The component to deploy. If this attribute refers to a file, it must be a valid archive. If this attribute refers to a directory, it must contain a valid archive in which all components have been exploded. If `upload` is set to `false`, this must be an absolute path on the server machine.
`name`	file name without extension	(optional) The display name for the component being deployed.
`type`	determined by extension	(optional) Deprecated.
`force`	`true`	(optional) If `true`, the component is overwritten if it already exists on the server. If `false`, `sun-appserv-deploy` fails if the component exists.
`retrievestubs`	client stubs not saved	(optional) The directory where client stubs are saved. This attribute is inherited by nested `component` elements.
`precompilejsp`	`false`	(optional) If `true`, all JSP files found in an enterprise application (`.ear`) or web application (`.war`) are precompiled. This attribute is ignored for other component types. This attribute is inherited by nested `component` elements.
`verify`	`false`	(optional) If `true`, syntax and semantics for all deployment descriptors are automatically verified for correctness. This attribute is inherited by nested `component` elements.
`contextroot`	file name without extension	(optional) The context root for a web module (WAR file). This attribute is ignored if the component is not a WAR file.

TABLE 3–6 `sun-appserv-deploy` Attributes *(Continued)*

Attribute	Default	Description
`dbvendorname`	`sun-ejb-jar.xml` entry	(optional) The name of the database vendor for which tables can be created. Allowed values are `db2`, `mssql`, `oracle`, `derby`, and `sybase`, case-insensitive. If not specified, the value of the `database-vendor-name` attribute in `sun-ejb-jar.xml` is used. If no value is specified, a connection is made to the resource specified by the `jndi-name` subelement of the `cmp-resource` element in the `sun-ejb-jar.xml` file, and the database vendor name is read. If the connection cannot be established, or if the value is not recognized, SQL-92 compliance is presumed. For details, see "Generation Options" on page 173.
`createtables`	`sun-ejb-jar.xml` entry	(optional) If `true`, causes database tables to be created for beans that need them. If `false`, does not create tables. If not specified, the value of the `create-tables-at-deploy` attribute in `sun-ejb-jar.xml` is used. For details, see "Generation Options" on page 173.

TABLE 3–6 `sun-appserv-deploy` Attributes *(Continued)*

Attribute	Default	Description
`dropandcreatetables`	`sun-ejb-jar.xml` entry	(optional) If `true`, and if tables were automatically created when this application was last deployed, tables from the earlier deployment are dropped and fresh ones are created. If `true`, and if tables were *not* automatically created when this application was last deployed, no attempt is made to drop any tables. If tables with the same names as those that would have been automatically created are found, the deployment proceeds, but a warning indicates that tables could not be created. If `false`, settings of `create-tables-at-deploy` or `drop-tables-at-undeploy` in the `sun-ejb-jar.xml` file are overridden. For details, see "Generation Options" on page 173.
`uniquetablenames`	`sun-ejb-jar.xml` entry	(optional) If `true`, specifies that table names are unique within each application server domain. If not specified, the value of the `use-unique-table-names` property in `sun-ejb-jar.xml` is used. For details, see "Generation Options" on page 173.
`enabled`	`true`	(optional) If `true`, enables the component.
`deploymentplan`	none	(optional) A deployment plan is a JAR file containing Sun-specific descriptors. Use this attribute when deploying an EAR file that lacks Sun-specific descriptors.
`upload`	`true`	(optional) If `true`, the component is transferred to the server for deployment. If the component is being deployed on the local machine, set upload to `false` to reduce deployment time. If a directory is specified for deployment, upload must be `false`.

TABLE 3–6 `sun-appserv-deploy` Attributes *(Continued)*

Attribute	Default	Description
`virtualservers`	default virtual server only	(optional) A comma-separated list of virtual servers to be deployment targets. This attribute applies only to application (`.ear`) or web (`.war`) components and is ignored for other component types.
`user`	`admin`	(optional) The user name used when logging into the application server.
`password`	none	(optional) Deprecated, use `passwordfile` instead. The password used when logging into the application server.
`passwordfile`	none	(optional) File containing passwords. The password from this file is retrieved for communication with the application server. If both `password` and `passwordfile` are specified, `passwordfile` takes precedence.
`host`	`localhost`	(optional) Target server. When deploying to a remote server, use the fully qualified host name.
`port`	`4848`	(optional) The administration port on the target server.
`asinstalldir`	see description	(optional) The installation directory for the local Application Server installation, which is used to find the administrative classes. If not specified, the command checks to see if the `asinstalldir` parameter has been set. Otherwise, administrative classes must be in the system classpath.
`sunonehome`	see description	(optional) Deprecated. Use `asinstalldir` instead.

Examples

Here is a simple application deployment script with many implied attributes:

```
<sun-appserv-deploy
file="${assemble}/simpleapp.ear"
passwordfile="${passwordfile}" />
```

Here is an equivalent script showing all the implied attributes:

```
<sun-appserv-deploy
 file="${assemble}/simpleapp.ear"
 name="simpleapp"
 force="true"
 precompilejsp="false"
 verify="false"
 upload="true"
 user="admin"
 passwordfile="${passwordfile}"
 host="localhost"
 port="4848"
 asinstalldir="${asinstalldir}" />
```

This example deploys multiple components to the same Application Server running on a remote server:

```
<sun-appserv-deploy passwordfile="${passwordfile}" host="greg.sun.com"
         asinstalldir="/opt/sun" >
 <component file="${assemble}/simpleapp.ear"/>
 <component file="${assemble}/simpleservlet.war"
     contextroot="test"/>
 <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

This example deploys the same components as the previous example because the three components match the `fileset` criteria, but note that it's not possible to set some component-specific attributes. All component-specific attributes (`name` and `contextroot`) use their default values.

```
<sun-appserv-deploy passwordfile="${passwordfile}" host="greg.sun.com"
         asinstalldir="/opt/sun" >
 <fileset dir="${assemble}" includes="**/*.?ar" />
</sun-appserv-deploy>
```

sun-appserv-undeploy

Undeploys any of the following.

- Enterprise application (EAR file)
- Web application (WAR file)
- Enterprise Java Bean (EJB-JAR file)
- Enterprise connector (RAR file)
- Application client

Subelements

The following table describes subelements for the `sun-appserv-undeploy` task. These are objects upon which this task acts.

TABLE 3–7 `sun-appserv-undeploy` Subelements

Element	Description
"component" on page 114	A component to be deployed.
"fileset" on page 116	A set of component files that match specified parameters.

Attributes

The following table describes attributes for the `sun-appserv-undeploy` task.

TABLE 3–8 `sun-appserv-undeploy` Attributes

Attribute	Default	Description
`name`	file name without extension	(optional if a `component` or `fileset` subelement is present or the `file` attribute is specified, otherwise required) The display name for the component being undeployed.
`file`	none	(optional) The component to undeploy. If this attribute refers to a file, it must be a valid archive. If this attribute refers to a directory, it must contain a valid archive in which all components have been exploded.
`type`	determined by extension	(optional) Deprecated.
`droptables`	`sun-ejb-jar.xml` entry	(optional) If `true`, causes database tables that were automatically created when the bean(s) were last deployed to be dropped when the bean(s) are undeployed. If `false`, does not drop tables. If not specified, the value of the `drop-tables-at-undeploy` attribute in `sun-ejb-jar.xml` is used. For details, see "Generation Options" on page 173.

TABLE 3–8 `sun-appserv-undeploy` Attributes *(Continued)*

Attribute	Default	Description
`cascade`	`false`	(optional) If `true`, deletes all connection pools and connector resources associated with the resource adapter being undeployed. If `false`, undeployment fails if any pools or resources are still associated with the resource adapter. This attribute is applicable to connectors (resource adapters) and applications with connector modules.
`user`	`admin`	(optional) The user name used when logging into the application server.
`password`	none	(optional) Deprecated, use `passwordfile` instead. The password used when logging into the application server.
`passwordfile`	none	(optional) File containing passwords. The password from this file is retrieved for communication with the application server. If both `password` and `passwordfile` are specified, `passwordfile` takes precedence.
`host`	`localhost`	(optional) Target server. When deploying to a remote server, use the fully qualified host name.
`port`	4848	(optional) The administration port on the target server.
`asinstalldir`	see description	(optional) The installation directory for the local Application Server installation, which is used to find the administrative classes. If not specified, the command checks to see if the `asinstalldir` parameter has been set. Otherwise, administrative classes must be in the system classpath.
`sunonehome`	see description	(optional) Deprecated. Use `asinstalldir` instead.

Examples

Here is a simple application undeployment script with many implied attributes:

```
<sun-appserv-undeploy name="simpleapp" passwordfile="${passwordfile}" />
```

Here is an equivalent script showing all the implied attributes:

```
<sun-appserv-undeploy
 name="simpleapp"
 user="admin"
 passwordfile="${passwordfile}"
 host="localhost"
 port="4848"
 asinstalldir="${asinstalldir}" />
```

This example demonstrates using the archive files (EAR and WAR, in this case) for the undeployment, using the component name (for undeploying the EJB component in this example), and undeploying multiple components.

```
<sun-appserv-undeploy passwordfile="${passwordfile}">
 <component file="${assemble}/simpleapp.ear"/>
 <component file="${assemble}/simpleservlet.war"/>
 <component name="simplebean" />
</sun-appserv-undeploy>
```

sun-appserv-component

Enables or disables the following J2EE component types that have been deployed to the Application Server.

- Enterprise application (EAR file)
- Web application (WAR file)
- Enterprise Java Bean (EJB-JAR file)
- Enterprise connector (RAR file)
- Application client

You don't need to specify the archive to enable or disable a component: only the component name is required. You can use the component archive, however, because it implies the component name.

Subelements

The following table describes subelements for the `sun-appserv-component` task. These are objects upon which this task acts.

TABLE 3–9 `sun-appserv-component` Subelements

Element	Description
"component" on page 114	A component to be deployed.
"fileset" on page 116	A set of component files that match specified parameters.

Attributes

The following table describes attributes for the `sun-appserv-component` task.

TABLE 3–10 `sun-appserv-component` Attributes

Attribute	Default	Description
`action`	none	The control command for the target application server. Valid values are `enable` and `disable`.
`name`	file name without extension	(optional if a `component` or `fileset` subelement is present or the `file` attribute is specified, otherwise required) The display name for the component being enabled or disabled.
`file`	none	(optional) The component to enable or disable. If this attribute refers to a file, it must be a valid archive. If this attribute refers to a directory, it must contain a valid archive in which all components have been exploded.
`type`	determined by extension	(optional) Deprecated.
`user`	`admin`	(optional) The user name used when logging into the application server.
`password`	none	(optional) Deprecated, use `passwordfile` instead. The password used when logging into the application server.
`passwordfile`	none	(optional) File containing passwords. The password from this file is retrieved for communication with the application server. If both `password` and `passwordfile` are specified, `passwordfile` takes precedence.
`host`	`localhost`	(optional) Target server. When enabling or disabling a remote server, use the fully qualified host name.
`port`	`4848`	(optional) The administration port on the target server.
`asinstalldir`	see description	(optional) The installation directory for the local Application Server installation, which is used to find the administrative classes. If not specified, the command checks to see if the `asinstalldir` parameter has been set. Otherwise, administrative classes must be in the system classpath.
`sunonehome`	see description	(optional) Deprecated. Use `asinstalldir` instead.

Examples

Here is a simple example of disabling a component:

```
<sun-appserv-component
 action="disable"
 name="simpleapp"
 passwordfile="${passwordfile}" />
```

Here is a simple example of enabling a component:

```
<sun-appserv-component
 action="enable"
 name="simpleapp"
 passwordfile="${passwordfile}" />
```

Here is an equivalent script showing all the implied attributes:

```
<sun-appserv-component
 action="enable"
 name="simpleapp"
 user="admin"
 passwordfile="${passwordfile}"
 host="localhost"
 port="4848"
 asinstalldir="${asinstalldir}" />
```

This example demonstrates disabling multiple components using the archive files (EAR and WAR, in this case) and using the component name (for an EJB component in this example).

```
<sun-appserv-component action="disable" passwordfile="${passwordfile}">
 <component file="${assemble}/simpleapp.ear"/>
 <component file="${assemble}/simpleservlet.war"/>
 <component name="simplebean" />
</sun-appserv-component>
```

sun-appserv-admin

Enables arbitrary administrative commands and scripts to be executed on the Application Server. This is useful for cases where a specific Ant task hasn't been developed or a set of related commands are in a single script.

Subelements

none

Attributes

The following table describes attributes for the `sun-appserv-admin` task.

TABLE 3–11 `sun-appserv-admin` Attributes

Attribute	Default	Description
`command`	none	(exactly one of these is required: `command`, `commandfile`, or `explicitcommand`) The command to execute. If the `user`, `passwordfile`, `host`, or `port` attributes are also specified, they are automatically inserted into the command before execution. If any of these options are specified in the command string, the corresponding attribute values are ignored.
`commandfile`	none	(exactly one of these is required: `command`, `commandfile`, or `explicitcommand`) Deprecated. The command script to execute. If `commandfile` is used, the values of all other attributes are ignored. Be sure to end the script referenced by `commandfile` with the `exit` command; if you omit `exit`, the Ant task might appear to hang after the command script is called.
`explicitcommand`	none	(exactly one of these is required: `command`, `commandfile`, or `explicitcommand`) The exact command to execute. No command processing is done, and all other attributes are ignored.
`user`	`admin`	(optional) The user name used when logging into the application server.
`password`	none	(optional) Deprecated, use `passwordfile` instead. The password used when logging into the application server.
`passwordfile`	none	(optional) File containing passwords. The password from this file is retrieved for communication with the application server. If both `password` and `passwordfile` are specified, `passwordfile` takes precedence.
`host`	`localhost`	(optional) Target server. If it is a remote server, use the fully qualified host name.
`port`	4848	(optional) The administration port on the target server.
`asinstalldir`	see description	(optional) The installation directory for the local Application Server installation, which is used to find the administrative classes. If not specified, the command checks to see if the `asinstalldir` parameter has been set. Otherwise, administrative classes must be in the system classpath.

TABLE 3–11 `sun-appserv-admin` Attributes *(Continued)*

Attribute	Default	Description
`sunonehome`	see description	(optional) Deprecated. Use `asinstalldir` instead.

Examples

Here is an example of executing the `create-jms-dest` command:

```
<sun-appserv-admin command="create-jms-dest --desttype topic">
```

Here is an example of using `commandfile` to execute the `create-jms-dest` command:

```
<sun-appserv-admin commandfile="create_jms_dest.txt" instance="development">
```

The `create_jms_dest.txt` file contains the following:

```
create-jms-dest --user admin --passwordfile "${passwordfile}" --host
localhost --port 4848 --desttype topic --target server1 simpleJmsDest
```

Here is an example of using `explicitcommand` to execute the `create-jms-dest` command:

```
<sun-appserv-admin command="create-jms-dest --user admin --passwordfile
"${passwordfile}" --host localhost --port 4848 --desttype topic
--target server1 simpleJmsDest">
```

sun-appserv-jspc

Precompiles JSP source code into Application Server compatible Java code for initial invocation by Application Server. Use this task to speed up access to JSP files or to check the syntax of JSP source code. You can feed the resulting Java code to the `javac` task to generate class files for the JSP files.

Subelements

none

Attributes

The following table describes attributes for the `sun-appserv-jspc` task.

TABLE 3–12 `sun-appserv-jspc` Attributes

Attribute	Default	Description
`destdir`		The destination directory for the generated Java source files.
`srcdir`		(exactly one of these is required: `srcdir` or `webapp`) The source directory where the JSP files are located.
`webapp`		(exactly one of these is required: `srcdir` or `webapp`) The directory containing the web application. All JSP files within the directory are recursively parsed. The base directory must have a `WEB-INF` subdirectory beneath it. When `webapp` is used, `sun-appserv-jspc` hands off all dependency checking to the compiler.
`verbose`	`2`	(optional) The verbosity integer to be passed to the compiler.
`classpath`		(optional) The classpath for running the JSP compiler.
`classpathref`		(optional) A reference to the JSP compiler classpath.
`uribase`	`/`	(optional) The URI context of relative URI references in the JSP files. If this context does not exist, it is derived from the location of the JSP file relative to the declared or derived value of `uriroot`. Only pages translated from an explicitly declared JSP file are affected.
`uriroot`	see description	(optional) The root directory of the web application, against which URI files are resolved. If this directory is not specified, the first JSP file is used to derive it: each parent directory of the first JSP file is searched for a `WEB-INF` directory, and the directory closest to the JSP file that has one is used. If no `WEB-INF` directory is found, the directory `sun-appserv-jspc` was called from is used. Only pages translated from an explicitly declared JSP file (including tag libraries) are affected.
`package`		(optional) The destination package for the generated Java classes.
`asinstalldir`	see description	(optional) The installation directory for the local Application Server installation, which is used to find the administrative classes. If not specified, the command checks to see if the `asinstalldir` parameter has been set. Otherwise, administrative classes must be in the system classpath.
`sunonehome`	see description	(optional) Deprecated. Use `asinstalldir` instead.

Example

The following example uses the `webapp` attribute to generate Java source files from JSP files. The `sun-appserv-jspc` task is immediately followed by a `javac` task, which compiles the generated Java files into class files. The `classpath` value in the `javac` task must be all on one line with no spaces.

```
<sun-appserv-jspc
 destdir="${assemble.war}/generated"
 webapp="${assemble.war}"
 classpath="${assemble.war}/WEB-INF/classes"
 asinstalldir="${asinstalldir}" />
<javac
 srcdir="${assemble.war}/WEB-INF/generated"
 destdir="${assemble.war}/WEB-INF/generated"
 debug="on"
 classpath="${assemble.war}/WEB-INF/classes:${asinstalldir}/lib/
   appserv-rt.jar:${asinstalldir}/lib/appserv-ext.jar">
 <include name="**/*.java"/>
</javac>
```

sun-appserv-update

Enables deployed applications (EAR files) and modules (EJB JAR, RAR, and WAR files) to be updated and reloaded for fast iterative development. This task copies modified class files, XML files, and other contents of the archive files to the appropriate subdirectory of the *domain-dir*`/applications` directory, then touches the `.reload` file to cause dynamic reloading to occur.

This is a local task and must be executed on the same machine as the application server.

Subelements

none

Attributes

The following table describes attributes for the `sun-appserv-update` task.

TABLE 3–13 `sun-appserv-update` Attributes

Attribute	Default	Description
`file`	none	The component to update, which must be a valid archive.

TABLE 3–13 `sun-appserv-update` Attributes *(Continued)*

Attribute	Default	Description
`domain`	`domain1`	(optional) The domain in which the application has been previously deployed.

Example

The following example updates the J2EE application `foo.ear`, which is deployed to the default domain, `domain1`.

```
<sun-appserv-update file="foo.ear"/>
```

Reusable Subelements

Reusable subelements of the Ant tasks for the Application Server are as follows. These are objects upon which the Ant tasks act.

- "component" on page 114
- "fileset" on page 116

component

Specifies a J2EE component. Allows a single task to act on multiple components. The `component` attributes override corresponding attributes in the parent task; therefore, the parent task attributes function as default values.

Subelements

none

Attributes

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

TABLE 3–14 `component` Attributes

Attribute	Default	Description
`file`	none	(optional if the parent task is "sun-appserv-undeploy" on page 104 or "sun-appserv-component" on page 107) The target component. If this attribute refers to a file, it must be a valid archive. If this attribute refers to a directory, it must contain a valid archive in which all components have been exploded. If `upload` is set to `false`, this must be an absolute path on the server machine.
`name`	file name without extension	(optional) The display name for the component.
`type`	determined by extension	(optional) Deprecated.
`force`	`true`	(applies to "sun-appserv-deploy" on page 99 only, optional) If `true`, the component is overwritten if it already exists on the server. If `false`, the containing element's operation fails if the component exists.
`precompilejsp`	`false`	(applies to "sun-appserv-deploy" on page 99 only, optional) If `true`, all JSP files found in an enterprise application (`.ear`) or web application (`.war`) are precompiled. This attribute is ignored for other component types.
`retrievestubs`	client stubs not saved	(applies to "sun-appserv-deploy" on page 99 only, optional) The directory where client stubs are saved.
`contextroot`	file name without extension	(applies to "sun-appserv-deploy" on page 99 only, optional) The context root for a web module (WAR file). This attribute is ignored if the component is not a WAR file.
`verify`	`false`	(applies to "sun-appserv-deploy" on page 99 only, optional) If `true`, syntax and semantics for all deployment descriptors is automatically verified for correctness.

Examples

You can deploy multiple components using a single task. This example deploys each component to the same Application Server running on a remote server.

```
<sun-appserv-deploy passwordfile="${passwordfile}" host="greg.sun.com"
     asinstalldir="/opt/s1as8" >
<component file="${assemble}/simpleapp.ear"/>
<component file="${assemble}/simpleservlet.war"
     contextroot="test"/>
```

```
<component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

You can also undeploy multiple components using a single task. This example demonstrates using the archive files (EAR and WAR, in this case) and the component name (for the EJB component).

```
<sun-appserv-undeploy passwordfile="${passwordfile}">
<component file="${assemble}/simpleapp.ear"/
<component file="${assemble}/simpleservlet.war"/>
<component name="simplebean" />
</sun-appserv-undeploy>
```

You can enable or disable multiple components. This example demonstrates disabling multiple components using the archive files (EAR and WAR, in this case) and the component name (for the EJB component).

```
<sun-appserv-component action="disable" passwordfile="${passwordfile}">
<component file="${assemble}/simpleapp.ear"/>
<component file="${assemble}/simpleservlet.war"/>
<component name="simplebean" />
</sun-appserv-component>
```

fileset

Selects component files that match specified parameters. When `fileset` is included as a subelement, the `name` and `contextroot` attributes of the containing element must use their default values for each file in the `fileset`. For more information, see `http://ant.apache.org/manual/CoreTypes/fileset.html`.

◆ ◆ ◆ CHAPTER 4

Debugging Applications

This chapter gives guidelines for debugging applications in the Sun Java System Application Server. It includes the following sections:

Enabling Debugging

When you enable debugging, you enable both local and remote debugging. To start the server in debug mode, use the `--debug` option as follows:

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

You can then attach to the server from the debugger at its default JPDA port, which is 9009. For example, for UNIX systems:

```
jdb -attach 9009
```

For Windows:

```
jdb -connect com.sun.jdi.SocketAttach:port=9009
```

Application Server debugging is based on the JPDA (Java Platform Debugger Architecture). For more information, see "JPDA Options" on page 118.

You can enable debugging even when the application server is started without the `--debug` option. This is useful if you start the application server from the Windows Start Menu or if you want to make sure that debugging is always turned on.

▼ To set the server to automatically start up in debug mode

1. **Select the JVM Settings component under the relevant configuration in the Administration Console.**
2. **Check the Debug Enabled box.**
3. **To specify a different port (from 9009, the default) to use when attaching the JVM to a debugger, specify** `address=`*port-number* **in the Debug Options field.**
4. **If you wish to add JPDA options, add any desired JPDA debugging options in Debug Options. See "JPDA Options" on page 118 .**

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

JPDA Options

The default JPDA options in Application Server are as follows:

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9009
```

For Windows, you can change `dt_socket` to `dt_shmem`.

If you substitute `suspend=y`, the JVM starts in suspended mode and stays suspended until a debugger attaches to it. This is helpful if you want to start debugging as soon as the JVM starts.

To specify a different port (from 9009, the default) to use when attaching the JVM to a debugger, specify `address=`*port-number*.

You can include additional options. A list of JPDA debugging options is available at `http://java.sun.com/products/jpda/doc/conninv.html#Invocation`.

Generating a Stack Trace for Debugging

You can generate a Java stack trace for debugging as described here if the Application Server is in verbose mode (see "Enabling Verbose Mode" on page 121):

http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/

The stack trace goes to the *domain-dir*`/logs/server.log` file and also appears on the command prompt screen.

If the `-Xrs` flag is set (for reduced signal usage) in the `domain.xml` file (under `jvm-options`), comment it out before generating the stack trace. If the `-Xrs` flag is used, the server might simply dump core and restart when you send the signal to generate the trace. For more about the `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

The Java Debugger

The Java Debugger (`jdb`) helps you find and fix bugs in Java language programs. When using the `jdb` debugger with the Application Server, you must attach to the server from the debugger at its default JPDA port, which is `9009`. For example, for UNIX systems:

```
jdb -attach 9009
```

For Windows:

```
jdb -connect com.sun.jdi.SocketAttach:port=9009
```

For more information about the `jdb` debugger, see the following links:

Java Platform Debugger Architecture - The Java Debugger:
`http://java.sun.com/products/jpda/doc/soljdb.html`

Java Platform Debugger Architecture - Connecting with JDB:
`http://java.sun.com/products/jpda/doc/conninv.html#JDB`

You can attach to the Application Server using any JPDA compliant debugger, including that of NetBeans (`http://www.netbeans.org`), Sun Java Studio, JBuilder, Eclipse, and so on.

Using an IDE

You can use an IDE (integrated development environment) with the Application Server to take advantage of the IDE's debugging features.

▼ To use the NetBeans IDE for Debugging

The following steps are applicable to the NetBeans 5 IDE and the Sun Java Studio 8 software, which is built on the NetBeans IDE.

1. **Download the latest version of NetBeans from `http://www.netbeans.org`.**

   This site also provides documentation for the NetBeans IDE.

2. **Start the NetBeans IDE.**

3. **If an Application Server is not already configured in the NetBeans IDE, perform the following steps:**

   a. **Select the Runtime tab to display the Runtime window.**

   b. **Right-click on Servers in the Runtime window.**

   c. **Select the Add Server command from the menu.**

   d. **On the first screen, select Sun Java System Application Server in the Server field, and type a name in the Name field. Select Next.**

   e. **On the second screen, fill in the requested configuration information. In the Domains folder field, use the Browse button to go to the Application Server *domain-root-dir* directory.**

   f. **Click Finish.**

4. **Create a project (an application or module) in the NetBeans IDE.**

5. **Right-click on the project in the component tree and select Debug Project from the menu.**

   This stops the Application Server and restarts it in debug mode.

6. **Set break points in your source file in the NetBeans IDE as usual, and run the application.**

Sun Java System Message Queue Debugging

Sun Java System Message Queue has a broker logger, which can be useful for debugging JMS, including message-driven bean, applications. You can adjust the logger's verbosity, and you can send the logger output to the broker's console using the broker's `-tty` option. For more information, see the *Sun Java System Message Queue 3.7 UR1 Administration Guide*.

Enabling Verbose Mode

If you want to see the server logs and messages printed to `System.out` on your command prompt screen, you can start the server in verbose mode. This makes it easy to do simple debugging using print statements, without having to view the `server.log` file every time.

When the server is in verbose mode, messages are logged to the console or terminal window in addition to the log file. In addition, pressing Ctrl-C stops the server and pressing Ctrl-\ (on UNIX platforms) or Ctrl-Break (on Windows platforms) prints a thread dump. On UNIX platforms, you can also print a thread dump using the `jstack` command (see `http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstack.html`) or the command `kill -QUIT` *process_id*.

To start the server in verbose mode, use the `--verbose` option as follows:

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

You can enable verbose mode even when the application server is started without the `--verbose` option. This is useful if you start the application server from the Windows Start Menu or if you want to make sure that verbose mode is always turned on.

You can set the server to automatically start up in verbose mode using the Administration Console. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Logging

You can use the Application Server's log files to help debug your applications. In the Administration Console, select the Application Server component, then click on the Open Log Viewer button in the General Information page. For details about logging, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Profiling

You can use a profiler to perform remote profiling on the Application Server to discover bottlenecks in server-side performance. This section describes how to configure these profilers for use with the Application Server:

Information about comprehensive monitoring and management support in the Java™ 2 Platform, Standard Edition (J2SE™ platform) version 5.0 is available at `http://java.sun.com/j2se/1.5.0/docs/guide/management/index.html`.

The HPROF Profiler

HPROF is a simple profiler agent shipped with the Java 2 SDK. It is a dynamically linked library that interacts with the JVMPI and writes out profiling information either to a file or to a socket in ASCII or binary format.

HPROF can present CPU usage, heap allocation statistics, and monitor contention profiles. In addition, it can also report complete heap dumps and states of all the monitors and threads in the Java virtual machine. For more details on the HPROF profiler, see the JDK documentation at `http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html#hprof`.

Once HPROF is enabled using the following instructions, its libraries are loaded into the server process.

▼ To use HPROF profiling on UNIX

1 Configure the Application Server using the Administration Console:

a. Select the JVM Settings component under the relevant configuration, then select the Profiler tab.

b. Edit the following fields:

- Profiler Name: `hprof`
- Profiler Enabled: `true`
- Classpath: (leave blank)
- Native Library Path: (leave blank)
- JVM Option: For each of these options, select Add, type the option in the Value field, then check its box:

  `-Xrunhprof:file=log.txt,`*options*

Here is an example of *options* you can use:

```
-Xrunhprof:file=log.txt,thread=y,depth=3
```

The `file` option determines where the stack dump is written in Step 2.

The syntax of HPROF options is as follows:

```
-Xrunhprof[:help]|[:option=value,option2=value2, ...]
```

Using `help` lists options that can be passed to HPROF. The output is as follows:

```
Hprof usage: -Xrunhprof[:help]|[:<option>=<value>, ...]

Option Name and Value   Description             Default
---------------------   -----------             -------
heap=dump|sites|all     heap profiling          all
cpu=samples|old         CPU usage               off
format=a|b              ascii or binary output  a
file=<file>             write data to file      java.hprof
                                (.txt for ascii)
net=<host>:<port>       send data over a socket write to file
depth=<size>            stack trace depth       4
cutoff=<value>          output cutoff point     0.0001
lineno=y|n              line number in traces?  y
thread=y|n              thread in traces?       n
doe=y|n                 dump on exit?           y
```

2 **Restart the Application Server. This writes an HPROF stack dump to the file you specified using the `file` HPROF option in Step 1.**

The Optimizeit Profiler

You can purchase Optimizeit™ from Borland at `http://www.borland.com/optimizeit`.

Once Optimizeit is enabled using the following instructions, its libraries are loaded into the server process.

▼ To enable remote profiling with Optimizeit

1 **Configure your operating system:**

- On Solaris, add *Optimizeit-dir*`/lib` to the LD_LIBRARY_PATH environment variable.
- On Windows, add *Optimizeit-dir*`/lib` to the PATH environment variable.

2. **Configure the Application Server using the Administration Console:**

   a. **Select the JVM Settings component under the relevant configuration, then select the Profiler tab.**

   b. **Edit the following fields:**

   - Profiler Name: `optimizeit`
   - Profiler Enabled: `true`
   - Classpath: *Optimizeit-dir*`/lib/optit.jar`
   - Native Library Path: *Optimizeit-dir*`/lib`
   - JVM Option: For each of these options, select Add, type the option in the Value field, then check its box:

     ```
     -DOPTITHOME=Optimizeit-dir  -Xrunpri:startAudit=t
     -Xbootclasspath/p:/Optimizeit-dir/lib/oibcp.jar
     ```

3. **In addition, you might have to set the following in your `server.policy` file.**

   For more information about the `server.policy` file, see "The server.policy File" on page 54

   ```
   grant codeBase "file:Optimizeit-dir/lib/optit.jar" {
       permission java.security.AllPermission;
   };
   ```

4. **Restart the Application Server.**

   When the server starts up with this configuration, you can attach the profiler.

See Also For further details, see the Optimizeit documentation.

Troubleshooting If any of the configuration options are missing or incorrect, the profiler might experience problems that affect the performance of the Application Server.

PART II

Developing Applications and Application Components

◆ ◆ ◆ CHAPTER 5

Developing Web Applications

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

For general information about web applications, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/WebApp.html#wp76431`.

Introducing Web Applications

This section includes summaries of the following topics:

Internationalization Issues

This section covers internationalization as it applies to the following:

The Server

To set the default locale of the entire Application Server, which determines the locale of the Administration Console, the logs, and so on, use the Administration Console. Select the Domain component, and type a value in the Locale field. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Servlets

This section explains how the Application Server determines the character encoding for the servlet request and the servlet response. For encodings you can use, see `http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html`.

Servlet Request

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

- The `getCharacterEncoding()` method.
- A hidden field in the form, specified by the `form-hint-field` attribute of the `parameter-encoding` element in the `sun-web.xml` file.
- The character encoding set in the `default-charset` attribute of the `parameter-encoding` element in the `sun-web.xml` file.
- The default, which is `ISO-8859-1`.

For details about the `parameter-encoding` element, see "parameter-encoding" on page 363.

Servlet Response

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

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

Virtual Servers

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

When you first install the Application Server, a default virtual server is created. (You can also assign a default virtual server to each new HTTP listener you create. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.)

Web applications and J2EE applications containing web components can be assigned to virtual servers.

▼ To assign virtual servers

1. **Deploy the application or web module and assign the desired virtual server to it.**
   For more information, see "Tools for Deployment" on page 93.
2. **In the Administration Console, open the HTTP Service component under the relevant configuration.**
3. **Open the Virtual Servers component under the HTTP Service component.**
4. **Select the virtual server to which you want to assign a default web module.**
5. **Select the application or web module from the Default Web Module drop-down list.**
   For more information, see "Default Web Modules" on page 129.

See Also For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Default Web Modules

A default web module can be assigned to the default virtual server and to each new virtual server. For details, see "Virtual Servers" on page 128. To access the default web module for a virtual server, point the browser to the URL for the virtual server, but do not supply a context root. For example:

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

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

For example:

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

Classloader Delegation

The Servlet specification recommends that the Web Classloader look in the local class loader before delegating to its parent. To make the Web Classloader follow the delegation model in the Servlet specification, set `delegate="false"` in the `class-loader` element of the `sun-web.xml` file. It's safe to do this only for a web module that does not interact with any other modules.

The default value is `delegate="true"`, which causes the Web Classloader to delegate in the same manner as the other classloaders. Use `delegate="true"` for a web application that accesses EJB components or that acts as a web service client or endpoint. For details about `sun-web.xml`, see "The sun-web.xml File" on page 285.

For general information about classloaders, see "Classloaders" on page 78.

Using the default-web.xml File

You can use the `default-web.xml` file to define features such as filters and security constraints that apply to all web applications.

▼ To use the default-web.xml file

1. **Place the JAR file for the filter, security constraint, or other feature in the *domain-dir*`/lib` directory.**
2. **Edit the *domain-dir*`/config/default-web.xml` file to refer to the JAR file.**
3. **Restart the server.**

More Information

The InvokerServlet

The `InvokerServlet` allows use of the `servlet-name` instead of the `servlet-mapping` for invoking a servlet with a URL, as described in "Invoking a Servlet with a URL" on page 133. The `InvokerServlet` is commented out in the `default-web.xml` file. To re-enable the `InvokerServlet`, remove the comment indicators (`<!--` and `-->`), then restart the server.

Configuring Logging in the Web Container

For information about configuring logging and monitoring in the web container using the Administration Console, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Configuring HTML Error Pages

To specify an error page (or URL to an error page) to be displayed to the end user, use the `error-url` attribute of the "sun-web-app" on page 398 element in the `sun-web.xml` file. For example:

```
<sun-web-app error-url="webserver-install-dir/error/error1.html">
... subelements ...
</sun-web-app>
```

For details, see "sun-web-app" on page 398.

If the `error-url` attribute is specified, it overrides all other mechanisms configured for error reporting.

Note – This attribute should not point to a URL on the Application Server instance, because the `error-url` cannot be loaded if the server is down. Instead, specify a URL that points to a location on the web server.

The Application Server provides the following options for specifying the error page.

- You can specify the `error-url` to be an HTTP URL. The Application Server forwards the client request to the specified error URL.
- If you do not specify the `error-url` attribute in the `sun-web.xml` file, a default error page is displayed.

The error page is displayed according to the following rules:

- When an error is encountered for an application, the Application Server first checks if the `error-url` attribute is defined. If it is defined, the Application Server reads the URL attribute and loads the error page.
- If the `error-url` attribute is missing or invalid, the Application Server displays the default error page.
- If the `error-url` has been defined but the page is missing, the Application Server loads the default error page.
- If the default error page is missing, the error is forwarded to the web server.

Header Management

In the Platform Edition of the Application Server, the `Enumeration` from `request.getHeaders()` contains multiple elements instead of a single, aggregated value.

Redirecting URLs

You can specify that a request for an old URL is treated as a request for a new URL. This is called *redirecting* a URL.

To specify a redirected URL for a virtual server, use the `redirect_n` property, where *n* is a positive integer that allows specification of more than one. This property is a subelement of a `virtual-server` element in the `domain.xml` file. For more information about this element, see `virtual-server` in *Sun Java System Application Server Platform Edition 8.2 Administration Reference*. Each of these `redirect_n` properties is inherited by all web applications deployed on the virtual server.

The value of each `redirect_n` property has two components, which may be specified in any order:

The first component, `from`, specifies the prefix of the requested URI to match.

The second component, `url-prefix`, specifies the new URL prefix to return to the client. The `from` prefix is simply replaced by this URL prefix.

For example:

```
<property name="redirect_1" value="from=/dummy url-prefix=http://etude"/>
```

Using Servlets

Application Server supports the Java Servlet Specification version 2.4.

Note – Servlet API version 2.4 is fully backward compatible with version 2.3, so all existing servlets should work without modification or recompilation.

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

The Application Server provides the `wscompile` and `wsdeploy` tools to help you implement a web service endpoint as a servlet. For more information about these tools, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

This section describes how to create effective servlets to control application interactions running on an Application Server, including standard-based servlets. In addition, this section describes the Application Server features to use to augment the standards.

This section contains the following topics:

Invoking a Servlet with a URL

You can call a servlet deployed to the Application Server by using a URL in a browser or embedded as a link in an HTML or JSP file. The format of a servlet invocation URL is as follows:

`http://`*server*`:`*port*`/`*context-root*`/`*servlet-mapping*`?`*name*`=`*value*

The following table describes each URL section.

TABLE 5–1 URL Fields for Servlets Within an Application

URL element	Description
server:*port*	The IP address (or host name) and optional port number. To access the default web module for a virtual server, specify only this URL section. You do not need to specify the *context-root* or *servlet-name* unless you also wish to specify name-value parameters.
context-root	For an application, the context root is defined in the `context-root` element of the `application.xml` or `sun-application.xml` file. For an individually deployed web module, the context root is specified during deployment. For both applications and individually deployed web modules, the default context root is the name of the WAR file minus the `.war` suffix.
servlet-mapping	The `servlet-mapping` as configured in the `web.xml` file. You can use the `servlet-name` instead if you enable the `InvokerServlet`; see "Using the default-web.xml File" on page 130.
?*name*=*value*. . .	Optional request parameters.

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

```
http://localhost:8080/MortPages/calcMortgage?rate=8.0&per=360&bal=180000
```

When invoking a servlet from within a JSP file, you can use a relative path. For example:

```
<jsp:forward page="TestServlet"/>
<jsp:include page="TestServlet"/>
```

Servlet Output

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

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

To change these defaults using the Administration Console, select the Logger Settings component under the relevant configuration, then check or uncheck these boxes:

- Log Messages to Standard Error - If checked, `System.err` output is sent to the server log. If unchecked, `System.err` output is sent to the system default location only.
- Write to System Log - If checked, `System.out` output is sent to the server log. If unchecked, `System.out` output is sent to the system default location only.

For more information, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Caching Servlet Results

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

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

To define how an Application Server web application handles response caching, you edit specific fields in the `sun-web.xml` file.

Note – A servlet that uses caching is not portable.

A sample caching application is in *install-dir*`/samples/webapps/apps/caching`.

For more information about JSP caching, see "JSP Caching" on page 139.

The rest of this section covers the following topics:

- "Caching Features" on page 135

Caching Features

The Application Server has the following web application response caching capabilities:

- Caching is configurable based on the servlet name or the URI.
- When caching is based on the URI, this includes user specified parameters in the query string. For example, a response from `/garden/catalog?category=roses` is different from a response from `/garden/catalog?category=lilies`. These responses are stored under different keys in the cache.
- Cache size, entry timeout, and other caching behaviors are configurable.
- Entry timeout is measured from the time an entry is created or refreshed. To override this timeout for an individual cache mapping, specify the `cache-mapping` subelement `timeout`.
- To determine caching criteria programmatically, write a class that implements the `com.sun.appserv.web.cache.CacheHelper` interface. For example, if only a servlet knows when a back end data source was last modified, you can write a helper class to retrieve the last modified timestamp from the data source and decide whether to cache the response based on that timestamp.
- To determine cache key generation programmatically, write a class that implements the `com.sun.appserv.web.cache.CacheKeyGenerator` interface. See "CacheKeyGenerator Interface" on page 137.
- All non-ASCII request parameter values specified in cache key elements must be URL encoded. The caching subsystem attempts to match the raw parameter values in the request query string.
- Since newly updated classes impact what gets cached, the web container clears the cache during dynamic deployment or reloading of classes.
- The following `HttpServletRequest` request attributes are exposed:
  - `com.sun.appserv.web.cachedServletName`, the cached servlet target
  - `com.sun.appserv.web.cachedURLPattern`, the URL pattern being cached
- Results produced by resources that are the target of a `RequestDispatcher.include()` or `RequestDispatcher.forward()` call are cached if caching has been enabled for those resources. For details, see the descriptions of the "cache-mapping" on page 308 and "dispatcher" on page 326 elements in the `sun-web.xml` file.

Default Cache Configuration

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

- The default cache timeout is 30 seconds.

- Only the HTTP GET method is eligible for caching.
- HTTP requests with cookies or sessions automatically disable caching.
- No special consideration is given to `Pragma:`, `Cache-control:`, or `Vary:` headers.
- The default key consists of the Servlet Path (minus `pathInfo` and the query string).
- A "least recently used" list is maintained to evict cache entries if the maximum cache size is exceeded.
- Key generation concatenates the servlet path with key field values, if any are specified.
- Results produced by resources that are the target of a `RequestDispatcher.include()` or `RequestDispatcher.forward()` call are never cached.

Caching Example

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

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

```
    <constraint-field name="category" scope="request.parameter>
        <value match-expr="equals" cache-on-match-failure="true">
       bogus
        </value>
    </constraint-field>
</cache-mapping>
<cache-mapping>
    <servlet-name> InfoServlet </servlet-name>
    <cache-helper-ref>myHelper</cache-helper-ref>
</cache-mapping>
</cache>
```

For more information about the `sun-web.xml` caching settings, see "cache" on page 304.

CacheKeyGenerator Interface

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

The name of the context attribute is configurable as the `value` of the `cacheKeyGeneratorAttrName` property in the `default-helper` element of the `sun-web.xml` deployment descriptor. For more information, see "default-helper" on page 324.

About the Servlet Engine

Servlets exist in and are managed by the servlet engine in the Application Server. The servlet engine is an internal object that handles all servlet meta functions. These functions include instantiation, initialization, destruction, access from other components, and configuration management. This section covers the following topics:

Instantiating and Removing Servlets

After the servlet engine instantiates the servlet, the servlet engine calls the servlet's `init()` method to perform any necessary initialization. You can override this method to perform an initialization function for the servlet's life, such as initializing a counter.

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

Request Handling

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

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

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

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

Using JavaServer Pages

The Application Server supports the following JSP features:

- JavaServer Pages (JSP) Specification version 2.0
- Precompilation of JSP files, which is especially useful for production servers
- JSP tag libraries and standard portable tags

For information about creating JSP files, see Sun Microsystem's JavaServer Pages web site at `http://java.sun.com/products/jsp/index.html`.

For information about Java Beans, see Sun Microsystem's JavaBeans web page at `http://java.sun.com/beans/index.html`.

This section describes how to use JavaServer Pages (JSP files) as page templates in an Application Server web application. This section contains the following topics:

- "JSP Tag Libraries and Standard Portable Tags" on page 139
- "JSP Caching" on page 139
- "Options for Compiling JSP Files" on page 142

JSP Tag Libraries and Standard Portable Tags

Application Server supports tag libraries and standard portable tags. For more information, see the JavaServer Pages Standard Tag Library (JSTL) page at `http://java.sun.com/products/jsp/jstl/index.jsp`.

Web applications don't need to bundle copies of the `jsf-impl.jar` or `appserv-jstl.jar` JSP tag libraries (in *install-dir*`/lib`) to use JavaServer™ Faces technology or JSTL, respectively. These tag libraries are automatically available to all web applications.

However, the *install-dir*`/lib/appserv-tags.jar` tag library for JSP caching is not automatically available to web applications. See "JSP Caching" on page 139, next.

JSP Caching

JSP caching lets you cache tag invocation results within the Java engine. Each can be cached using different cache criteria. For example, suppose you have invocations to view stock quotes, weather information, and so on. The stock quote result can be cached for 10 minutes, the weather report result for 30 minutes, and so on.

For more information about response caching as it pertains to servlets, see "Caching Servlet Results" on page 134.

JSP caching is implemented by a tag library packaged into the *install-dir*`/lib/appserv-tags.jar` file, which you can copy into the `WEB-INF/lib` directory of your web application. The `appserv-tags.tld` tag library descriptor file is in the `META-INF` directory of this JAR file.

Note – Web applications that use this tag library are not portable.

To allow all web applications to share this tag library, change the following elements in the `domain.xml` file. Change this:

```
<jvm-options>
-Dcom.sun.enterprise.taglibs=appserv-jstl.jar,jsf-impl.jar
</jvm-options>
```

to this:

```
<jvm-options>
-Dcom.sun.enterprise.taglibs=appserv-jstl.jar,jsf-impl.jar,appserv-tags.jar
</jvm-options>
```

and this:

```
<jvm-options>
-Dcom.sun.enterprise.taglisteners=jsf-impl.jar
</jvm-options>
```

to this:

```
<jvm-options>
-Dcom.sun.enterprise.taglisteners=jsf-impl.jar,appserv-tags.jar
</jvm-options>
```

For more information about the `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Refer to these tags in JSP files as follows:

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

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

The tags are as follows:

- "cache" on page 140
- "flush" on page 141

cache

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

Attributes

The following table describes attributes for the `cache` tag.

TABLE 5–2 `cache` Attributes

Attribute	Default	Description
`key`	*ServletPath_Suffix*	(optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry. If no key is specified, a number is generated according to the position of the tag in the page.

TABLE 5–2 cache Attributes *(Continued)*

Attribute	Default	Description
`timeout`	`60s`	(optional) The time in seconds after which the body of the tag is executed and the cache is refreshed. By default, this value is interpreted in seconds. To specify a different unit of time, add a suffix to the timeout value as follows: `s` for seconds, `m` for minutes, `h` for hours, `d` for days. For example, `2h` specifies two hours.
`nocache`	`false`	(optional) If set to `true`, the body content is executed and served as if there were no `cache` tag. This offers a way to programmatically decide whether the cached response is sent or whether the body has to be executed, though the response is not cached.
`refresh`	`false`	(optional) If set to `true`, the body content is executed and the response is cached again. This lets you programmatically refresh the cache immediately regardless of the `timeout` setting.

Example

The following example represents a cached JSP file:

```
<%@ taglib prefix="mypfx" uri="Sun ONE Application Server Tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<mypfx:cache                key="${sessionScope.loginId}"
            nocache="${param.nocache}"
            refresh="${param.refresh}"
            timeout="10m">
<c:choose>
    <c:when test="${param.page == 'frontPage'}">
        <%-- get headlines from database --%>
    </c:when>
    <c:otherwise>
        ...
    </c:otherwise>
</c:choose>
</mypfx:cache>
<mypfx:cache timeout="1h">
<h2> Local News </h2>
    <%-- get the headline news and cache them --%>
</mypfx:cache>
```

flush

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

Attributes

The following table describes attributes for the `flush` tag.

TABLE 5–3 `flush` Attributes

Attribute	Default	Description
`key`	*ServletPath_Suffix*	(optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry. If no key is specified, a number is generated according to the position of the tag in the page.

Examples

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

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

To flush the entire cache:

```
<c:if test="${empty sessionScope.clearCache}">
   <mypfx:flush />
</c:if>
```

Options for Compiling JSP Files

Application Server provides the following ways of compiling JSP 2.0 compliant source files into servlets:

- JSP files are automatically compiled at runtime.
- The `asadmin deploy` command has a `precompilejsp` option. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.
- The `sun-appserv-jspc` Ant task allows you to precompile JSP files; see "sun-appserv-jspc" on page 111.
- The `jspc` command line tool allows you to precompile JSP files at the command line. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Creating and Managing HTTP Sessions

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

This chapter contains the following sections:

- "Configuring Sessions" on page 143

- "Session Managers" on page 143

Configuring Sessions

This section covers the following topics:

- "Sessions, Cookies, and URL Rewriting" on page 143
- "Coordinating Session Access" on page 143

Sessions, Cookies, and URL Rewriting

To configure whether and how sessions use cookies and URL rewriting, edit the `session-properties` and `cookie-properties` elements in the `sun-web.xml` file for an individual web application. See "session-properties" on page 391 and "cookie-properties" on page 322 for more about the properties you can configure.

For information about configuring default session properties for the entire web container, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Coordinating Session Access

Make sure that multiple threads don't simultaneously modify the same session object in conflicting ways.

This is especially likely to occur in web applications that use HTML frames where multiple servlets are executing simultaneously on behalf of the same client. A good solution is to ensure that one of the servlets modifies the session and the others have read-only access.

Session Managers

A session manager automatically creates new session objects whenever a new session starts. In some circumstances, clients do not join the session, for example, if the session manager uses cookies and the client does not accept cookies.

Application Server offers these session management options, determined by the "session-manager" on page 390 element's `persistence-type` attribute in the `sun-web.xml` file:

- "The memory Persistence Type" on page 144, the default
- "The file Persistence Type" on page 144, which uses a file to store session data

---

Note – If the session manager configuration contains an error, the error is written to the server log and the default (`memory`) configuration is used.

The memory Persistence Type

This persistence type is not designed for a production environment that requires session persistence. It provides no session persistence. However, you can configure it so that the session state in memory is written to the file system prior to server shutdown.

To specify the `memory` persistence type for the entire web container, use the `configure-ha-persistence` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

To specify the `memory` persistence type for a specific web application, edit the `sun-web.xml` file as in the following example. The `persistence-type` property is optional, but must be set to `memory` if included. This overrides the web container availability settings for the web application.

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

The only manager property that the `memory` persistence type supports is sessionFilename, which is listed under "manager-properties" on page 351.

For more information about the `sun-web.xml` file, see "The sun-web.xml File" on page 285.

The file Persistence Type

This persistence type provides session persistence to the local file system, and allows a single server domain to recover the session state after a failure and restart. The session state is persisted in the background, and the rate at which this occurs is configurable. The store also provides passivation and activation of the session state to help control the amount of memory used. This option is not supported in a production environment. However, it is useful for a development system with a single server instance.

Note – Make sure the `delete` option is set in the `server.policy` file, or expired file-based sessions might not be deleted properly. For more information about `server.policy`, see "The server.policy File" on page 54.

To specify the `file` persistence type for the entire web container, use the `configure-ha-persistence` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

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

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

The `file` persistence type supports all the manager properties listed under "manager-properties" on page 351 except `sessionFilename`, and supports the `directory` store property listed under "store-properties" on page 393.

For more information about the `sun-web.xml` file, see "The sun-web.xml File" on page 285.

◆ ◆ ◆ CHAPTER 6

Using Enterprise JavaBeans Technology

This chapter describes how Enterprise JavaBeans™ (EJB™) technology is supported in the Sun Java System Application Server. This chapter addresses the following topics:

Summary of EJB 2.1 Changes

The Application Server supports the Sun Microsystems Enterprise JavaBeans (EJB) architecture as defined by the Enterprise JavaBeans Specification, v2.1 and is compliant with the Enterprise JavaBeans Specification, v2.0.

Note – The Application Server is backward compatible with 1.1 and 2.0 enterprise beans. However, to take advantage of version 2.1 features, you should develop new beans as 2.1 enterprise beans.

The changes in the Enterprise JavaBeans Specification, v2.1 that impact enterprise beans in the Application Server environment are as follows:

- **EJB Timer Service:** This is a container-managed, reliable, and transactional notification service that provides methods to allow callbacks to be scheduled for time-based events. See "EJB Timer Service" on page 151.

- **Message-driven beans:** This type of enterprise bean can consume any inbound messages from a Connector 1.5 inbound resource adapter, primarily but not exclusively JMS messages. See "Using Message-Driven Beans" on page 157.
- **EJB Web Services:** A stateless session bean can serve as a web service endpoint. In addition, all EJB component types can act as web service clients. For details, see the web service elements in the `sun-ejb-jar.xml` file, described in "The sun-ejb-jar.xml File" on page 288.

Value Added Features

The Application Server provides a number of value additions that relate to EJB development. These capabilities are discussed in the following sections (references to more in-depth material are included):

Read-Only Beans

Another feature that the Application Server provides is the *read-only bean*, an entity bean that is never modified by an EJB client. Read-only beans avoid database updates completely. A read-only bean is not portable.

A read-only bean can be used to cache a database entry that is frequently accessed but rarely updated (externally by other beans). When the data that is cached by a read-only bean is updated by another bean, the read-only bean can be notified to refresh its cached data.

The Application Server provides a number of ways by which a read-only bean's state can be refreshed. By setting the `refresh-period-in-seconds` element in the `sun-ejb-jar.xml` file and the `trans-attribute` element in the `ejb-jar.xml` file, it is easy to configure a read-only bean that is (a) always refreshed, (b) periodically refreshed, (c) never refreshed, or (d) programmatically refreshed.

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently. For further information and usage guidelines, see "Using Read-Only Beans" on page 154.

pass-by-reference

The `pass-by-reference` element in the `sun-ejb-jar.xml` file allows you to specify the parameter passing semantics for colocated remote EJB invocations. This is an opportunity to improve performance. However, use of this feature results in non-portable applications. See "pass-by-reference" on page 364.

Pooling and Caching

The EJB container of the Application Server pools anonymous instances (message-driven beans, stateless session beans, and entity beans) to reduce the overhead of creating and destroying objects. The EJB container maintains the free pool for each bean that is deployed. Bean instances in the free pool have no identity (that is, no primary key associated) and are used to serve the method calls of the home interface. The free beans are also used to serve all methods for stateless session beans.

Bean instances in the free pool transition from a Pooled state to a Cached state after `ejbCreate` and the business methods run. The size and behavior of each pool is controlled using pool-related properties in the EJB container or the `sun-ejb-jar.xml` file.

In addition, the Application Server supports a number of tunable parameters that can control the number of "stateful" instances (stateful session beans and entity beans) cached as well as the duration they are cached. Multiple bean instances that refer to the same database row in a table can be cached. The EJB container maintains a cache for each bean that is deployed.

To achieve scalability, the container selectively evicts some bean instances from the cache, usually when cache overflows. These evicted bean instances return to the free bean pool. The size and behavior of each cache can be controlled using the cache-related properties in the EJB container or the `sun-ejb-jar.xml` file.

Pooling and caching parameters for the `sun-ejb-jar.xml` file are described in "bean-cache" on page 302.

Pooling Parameters

One of the most important parameters of Application Server pooling is `steady-pool-size`. When `steady-pool-size` is set to greater than 0, the container not only pre-populates the bean pool with the specified number of beans, but also attempts to ensure that there is always this many beans in the free pool. This ensures that there are enough beans in the ready to serve state to process user requests.

This parameter does not necessarily guarantee that no more than `steady-pool-size` instances exist at a given time. It only governs the number of instances that are pooled over a long period of time. For example, suppose an idle stateless session container has a fully-populated pool with a `steady-pool-size` of `10`. If 20 concurrent requests arrive for the EJB component, the

container creates 10 additional instances to satisfy the burst of requests. The advantage of this is that it prevents the container from blocking any of the incoming requests. However, if the activity dies down to 10 or fewer concurrent requests, the additional 10 instances are discarded.

Another parameter, `pool-idle-timeout-in-seconds`, allows the administrator to specify, through the amount of time a bean instance can be idle in the pool. When `pool-idle-timeout-in-seconds` is set to greater than 0, the container removes or destroys any bean instance that is idle for this specified duration.

Caching Parameters

Application Server provides a way that completely avoids caching of entity beans, using commit option C. Commit option C is particularly useful if beans are accessed in large number but very rarely reused. For additional information, refer to "Commit Options" on page 163.

The Application Server caches can be either bounded or unbounded. *Bounded caches* have limits on the number of beans that they can hold beyond which beans are passivated. For stateful session beans, there are three ways (LRU, NRU and FIFO) of picking victim beans when cache overflow occurs. Caches can also passivate beans that are idle (not accessed for a specified duration).

Bean-Level Container-Managed Transaction Timeouts

The default transaction timeout for the domain is specified using the Transaction Timeout setting of the Transaction Service. A transaction started by the container must commit (or rollback) within this time, regardless of whether the transaction is suspended (and resumed), or the transaction is marked for rollback.

To override this timeout for an individual bean, use the optional `cmt-timeout-in-seconds` element in `sun-ejb-jar.xml`. The default value, `0`, specifies that the default Transaction Service timeout is used. The value of `cmt-timeout-in-seconds` is used for all methods in the bean that start a new container-managed transaction. This value is *not* used if the bean joins a client transaction.

Priority Based Scheduling of Remote Bean Invocations

You can create multiple thread pools, each having its own work queues. An optional element in the `sun-ejb-jar.xml` file, `use-thread-pool-id`, specifies the thread pool that processes the requests for the bean. The bean must have a remote interface, or `use-thread-pool-id` is ignored. You can create different thread pools and specify the appropriate thread pool ID for a bean that requires a quick response time. If there is no such thread pool configured or if the element is absent, the default thread pool is used.

Immediate Flushing

Normally, all entity bean updates within a transaction are batched and executed at the end of the transaction. The only exception is the database flush that precedes execution of a finder or select query.

Since a transaction often spans many method calls, you might want to find out if the updates made by a method succeeded or failed immediately after method execution. To force a flush at the end of a method's execution, use the "flush-at-end-of-method" on page 337 element in the `sun-ejb-jar.xml` file. Only non-finder methods in the Local, Local Home, Remote, and Remote Home interfaces of an entity bean can be flush-enabled.

Upon completion of the method, the EJB container updates the database. Any exception thrown by the underlying data store is wrapped as follows:

- If the method that triggered the flush is a `create` method, the exception is wrapped with `CreateException`.
- If the method that triggered the flush is a `remove` method, the exception is wrapped with `RemoveException`.
- For all other methods, the exception is wrapped with `EJBException`.

All normal end-of-transaction database synchronization steps occur regardless of whether the database has been flushed during the transaction.

EJB Timer Service

The EJB Timer Service uses a database to store persistent information about EJB timers. By default, the EJB Timer Service in Application Server is preconfigured to use an embedded version of Derby. The EJB Timer Service configuration can store persistent timer information in any database supported by the Application Server CMP container.

For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

To change the database used by the EJB Timer Service, set the EJB Timer Service's Timer DataSource setting to a valid JDBC resource. You must also create the timer database table. DDL files are located in *install-dir*`/lib/install/databases`.

Using the EJB Timer Service is equivalent to interacting with a single JDBC resource manager. If an EJB component or application accesses a database either directly through JDBC or indirectly (for example, through an entity bean's persistence mechanism), and also interacts with the EJB Timer Service, its data source must be configured with an XA JDBC driver.

You can change the following EJB Timer Service settings. You must restart the server for the changes to take effect.

- Minimum Delivery Interval - Specifies the minimum time in milliseconds before an expiration for a particular timer can occur. This guards against extremely small timer increments that can overload the server. The default is `7000`.
- Maximum Redeliveries - Specifies the maximum number of times the EJB timer service attempts to redeliver a timer expiration due for exception or rollback. The default is 1.
- Redelivery Interval - Specifies how long in milliseconds the EJB timer service waits after a failed `ejbTimeout` delivery before attempting a redelivery. The default is `5000`.
- Timer DataSource - Specifies the database used by the EJB Timer Service. The default is `jdbc/__TimerPool`.

For information about configuring EJB Timer Service settings, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*. For information about the `asadmin list-timers` command, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Using Session Beans

This section provides guidelines for creating session beans in the Application Server environment. This section addresses the following topics:

- "About the Session Bean Containers" on page 152
- "Restrictions and Optimizations" on page 154

Extensive information on session beans is contained in the chapters 6, 7, and 8 of the Enterprise JavaBeans Specification, v2.1.

About the Session Bean Containers

Like an entity bean, a session bean can access a database through Java™ Database Connectivity (JDBC™) calls. A session bean can also provide transaction settings. These transaction settings and JDBC calls are referenced by the session bean's container, allowing it to participate in transactions managed by the container.

A container managing stateless session beans has a different charter from a container managing stateful session beans.

Stateless Container

The *stateless container* manages stateless session beans, which, by definition, do not carry client-specific states. All session beans (of a particular type) are considered equal.

A stateless session bean container uses a bean pool to service requests. The Application Server specific deployment descriptor file, `sun-ejb-jar.xml`, contains the properties that define the pool:

- `steady-pool-size`
- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

For more information about `sun-ejb-jar.xml`, see "The sun-ejb-jar.xml File" on page 288.

The Application Server provides the `wscompile` and `wsdeploy` tools to help you implement a web service endpoint as a stateless session bean. For more information about these tools, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Stateful Container

The *stateful container* manages the stateful session beans, which, by definition, carry the client-specific state. There is a one-to-one relationship between the client and the stateful session beans. At creation, each stateful session bean (SFSB) is given a unique session ID that is used to access the session bean so that an instance of a stateful session bean is accessed by a single client only.

Stateful session beans are managed using cache. The size and behavior of stateful session beans cache are controlled by specifying the following `sun-ejb-jar.xml` parameters:

- `max-cache-size`
- `resize-quantity`
- `cache-idle-timeout-in-seconds`
- `removal-timeout-in-seconds`
- `victim-selection-policy`

The `max-cache-size` element specifies the maximum number of session beans that are held in cache. If the cache overflows (when the number of beans exceeds `max-cache-size`), the container then passivates some beans or writes out the serialized state of the bean into a file. The directory in which the file is created is obtained from the EJB container using the configuration APIs.

For more information about `sun-ejb-jar.xml`, see "The sun-ejb-jar.xml File" on page 288.

The passivated beans are stored on the file system. The Session Store Location setting in the EJB container allows the administrator to specify the directory where passivated beans are stored. By default, passivated stateful session beans are stored in application-specific subdirectories created under *domain-dir*`/session-store`.

Note – Make sure the `delete` option is set in the `server.policy` file, or expired file-based sessions might not be deleted properly. For more information about `server.policy`, see "The server.policy File" on page 54.

Restrictions and Optimizations

This section discusses restrictions on developing session beans and provides some optimization guidelines:

Optimizing Session Bean Performance

For stateful session beans, colocating the stateful beans with their clients so that the client and bean are executing in the same process address space improves performance.

Restricting Transactions

The following restrictions on transactions are enforced by the container and must be observed as session beans are developed:

- A session bean can participate in, at most, a single transaction at a time.
- If a session bean is participating in a transaction, a client cannot invoke a method on the bean such that the `trans-attribute` element in the `ejb-jar.xml` file would cause the container to execute the method in a different or unspecified transaction context or an exception is thrown.
- If a session bean instance is participating in a transaction, a client cannot invoke the `remove` method on the session object's home or component interface object or an exception is thrown.

Using Read-Only Beans

A *read-only bean* is an entity bean that is never modified by an EJB client. The data that a read-only bean represents can be updated externally by other enterprise beans, or by other means, such as direct database updates.

Note – Read-only beans are specific to Application Server and are not part of the Enterprise JavaBeans Specification, v2.1. Use of this feature results in a non-portable application.

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently. The following topics are addressed in this section:

Read-Only Bean Characteristics and Life Cycle

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently. For example, a read-only bean can be used to represent a stock quote for a particular company, which is updated externally. In such a case, using a regular entity bean might incur the burden of calling `ejbStore`, which can be avoided by using a read-only bean.

Read-only beans have the following characteristics:

- Only entity beans can be read-only beans.
- Either bean-managed persistence (BMP) or container-managed persistence (CMP) is allowed. If CMP is used, do not create the database schema during deployment. Instead, work with your database administrator to populate the data into the tables. See Chapter 7, "Using Container-Managed Persistence for Entity Beans."
- Only container-managed transactions are allowed; read-only beans cannot start their own transactions.
- Read-only beans don't update any bean state.
- `ejbStore` is never called by the container.
- `ejbLoad` is called only when a transactional method is called or when the bean is initially created (in the cache), or at regular intervals controlled by the bean's `refresh-period-in-seconds` element in the `sun-ejb-jar.xml` file.
- The home interface can have any number of find methods. The return type of the find methods must be the primary key for the same bean type (or a collection of primary keys).
- If the data that the bean represents can change, then `refresh-period-in-seconds` must be set to refresh the beans at regular intervals. `ejbLoad` is called at this regular interval.

A read-only bean comes into existence using the appropriate find methods.

Read-only beans are cached and have the same cache properties as entity beans. When a read-only bean is selected as a victim to make room in the cache, `ejbPassivate` is called and the bean is returned to the free pool. When in the free pool, the bean has no identity and is used only to serve any finder requests.

Read-only beans are bound to the naming service like regular read-write entity beans, and clients can look up read-only beans the same way read-write entity beans are looked up.

Read-Only Bean Good Practices

For best results, follow these guidelines when developing read-only beans:

- Avoid having any `create` or `remove` methods in the home interface.
- Use any of the valid EJB 2.1 transaction attributes for the `trans-attribute` element.

The reason for having `TX_SUPPORTED` is to allow reading uncommitted data in the same transaction. Also, the transaction attributes can be used to force `ejbLoad`.

Refreshing Read-Only Beans

There are several ways of refreshing read-only beans as addressed in the following sections:

- "Invoking a Transactional Method" on page 156
- "Refreshing Periodically" on page 156
- "Refreshing Programmatically" on page 156

Invoking a Transactional Method

Invoking any transactional method invokes `ejbLoad`.

Refreshing Periodically

Use the `refresh-period-in-seconds` element in the `sun-ejb-jar.xml` file to refresh a read-only bean periodically.

- If the value specified in `refresh-period-in-seconds` is zero or not specified, which is the default, the bean is never refreshed (unless a transactional method is accessed).
- If the value is greater than zero, the bean is refreshed at the rate specified.

Note – This is the only way to refresh the bean state if the data can be modified external to the Application Server.

Refreshing Programmatically

Typically, beans that update any data that is cached by read-only beans need to notify the read-only beans to refresh their state. Use `ReadOnlyBeanNotifier` to force the refresh of read-only beans.

To do this, invoke the following methods on the `ReadOnlyBeanNotifier` bean:

```
public interface ReadOnlyBeanNotifier extends java.rmi.Remote {
   refresh(Object PrimaryKey) throws RemoteException;
 }
```

The implementation of the `ReadOnlyBeanNotifier` interface is provided by the container. The bean looks up `ReadOnlyBeanNotifier` using a fragment of code such as the following example:

```
com.sun.appserv.ejb.ReadOnlyBeanHelper helper =
  new com.sun.appserv.ejb.ReadOnlyBeanHelper();
com.sun.appserv.ejb.ReadOnlyBeanNotifier notifier =
```

```
  helper.getReadOnlyBeanNotifier("java:comp/env/ejb/ReadOnlyCustomer");
notifier.refresh(PrimaryKey);
```

For a local read-only bean notifier, the lookup has this modification:

```
helper.getReadOnlyBeanLocalNotifier("java:comp/env/ejb/LocalReadOnlyCustomer");
```

Beans that update any data that is cached by read-only beans need to call the `refresh` methods. The next (non-transactional) call to the read-only bean invokes `ejbLoad`.

Note – Programmatic refresh of read-only beans is not supported in a clustered environment.

Deploying Read Only Beans

Read-only beans are deployed in the same manner as other entity beans. However, in the entry for the bean in the `sun-ejb-jar.xml` file, the `is-read-only-bean` element must be set to true. That is:

```
<is-read-only-bean>true</is-read-only-bean>
```

Also, the `refresh-period-in-seconds` element in the `sun-ejb-jar.xml` file can be set to some value that specifies the rate at which the bean is refreshed. If this element is missing, no refresh occurs.

All requests in the same transaction context are routed to the same read-only bean instance. Set the `allow-concurrent-access` element to either `true` (to allow concurrent accesses) or `false` (to serialize concurrent access to the same read-only bean). The default is `false`.

For further information on these elements, refer to "The sun-ejb-jar.xml File" on page 288.

Using Message-Driven Beans

This section describes message-driven beans and explains the requirements for creating them in the Application Server environment. This section contains the following topics:

Message-Driven Bean Configuration

This section addresses the following configuration topics:

■ "Domain-Level Settings" on page 158

Connection Factory and Destination

A message-driven bean is a client to a Connector 1.5 inbound resource adapter. The message-driven bean container uses the JMS service integrated into the Application Server for message-driven beans that are JMS clients. JMS clients use JMS Connection Factory- and Destination-administered objects. A JMS Connection Factory administered object is a resource manager Connection Factory object that is used to create connections to the JMS provider.

The `mdb-connection-factory` element in the `sun-ejb-jar.xml`file for a message-driven bean specifies the connection factory that creates the container connection to the JMS provider.

The `jndi-name` element of the `ejb` element in the `sun-ejb-jar.xml` file specifies the JNDI name of the administered object for the JMS `Queue` or `Topic` destination that is associated with the message-driven bean.

Message-Driven Bean Pool

The container manages a pool of message-driven beans for the concurrent processing of a stream of messages. The `sun-ejb-jar.xml` file contains the elements that define the pool (that is, the `bean-pool` element):

- `steady-pool-size`
- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

For more information about `sun-ejb-jar.xml`, see "The sun-ejb-jar.xml File" on page 288.

Domain-Level Settings

You can control the following domain-level message-driven bean settings in the EJB container:

- Initial and Minimum Pool Size - Specifies the initial and minimum number of beans maintained in the pool. The default is `0`.
- Maximum Pool Size - Specifies the maximum number of beans that can be created to satisfy client requests. The default is 32.
- Pool Resize Quantity - Specifies the number of beans to be created if a request arrives when the pool is empty (subject to the Initial and Minimum Pool Size), or the number of beans to remove if idle for more than the Idle Timeout. The default is `8`.
- Idle Timeout - Specifies the maximum time in seconds that a bean can remain idle in the pool. After this amount of time, the bean is destroyed. The default is `600` (10 minutes). A value of `0` means a bean can remain idle indefinitely.

For information on monitoring message-driven beans, see the Application Server Administration Console online help and the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Note – Running monitoring when it is not needed might impact performance, so you might choose to turn monitoring off when it is not in use. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Restrictions and Optimizations

This section discusses the following restrictions and performance optimizations that pertain to developing message-driven beans:

Pool Tuning and Monitoring

The message-driven bean pool is also a pool of threads, with each message-driven bean instance in the pool associating with a server session, and each server session associating with a thread. Therefore, a large pool size also means a high number of threads, which impacts performance and server resources.

When configuring message-driven bean pool properties, make sure to consider factors such as message arrival rate and pattern, `onMessage` method processing time, overall server resources (threads, memory, and so on), and any concurrency requirements and limitations from other resources that the message-driven bean accesses.

When tuning performance and resource usage, make sure to consider potential JMS provider properties for the connection factory used by the container (the `mdb-connection-factory` element in the `sun-ejb-jar.xml` file). For example, you can tune the Sun Java System Message Queue flow control related properties for connection factory in situations where the message incoming rate is much higher than `max-pool-size` can handle.

Refer to the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* for information on how to get message-driven bean pool statistics.

onMessage Runtime Exception

Message-driven beans, like other well-behaved MessageListeners, should not, in general, throw runtime exceptions. If a message-driven bean's `onMessage` method encounters a system-level exception or error that does not allow the method to successfully complete, the Enterprise JavaBeans Specification, v2.1 provides the following guidelines:

- If the bean method encounters a runtime exception or error, it should simply propagate the error from the bean method to the container.

- If the bean method performs an operation that results in a checked exception that the bean method cannot recover, the bean method should throw the `javax.ejb.EJBException` that wraps the original exception.
- Any other unexpected error conditions should be reported using `javax.ejb.EJBException` (`javax.ejb.EJBException` is a subclass of `java.lang.RuntimeException`).

Under container-managed transaction demarcation, upon receiving a runtime exception from a message-driven bean's `onMessage` method, the container rolls back the container-started transaction and the message is redelivered. This is because the message delivery itself is part of the container-started transaction. By default, the Application Server container closes the container's connection to the JMS provider when the first runtime exception is received from a message-driven bean instance's `onMessage` method. This avoids potential message redelivery looping and protects server resources if the message-driven bean's `onMessage` method continues misbehaving. To change this default container behavior, use the `cmt-max-runtime-exceptions` property of the `mdb-container` element in the `domain.xml` file.

The `cmt-max-runtime-exceptions` property specifies the maximum number of runtime exceptions allowed from a message-driven bean's `onMessage` method before the container starts to close the container's connection to the message source. By default this value is 1; -1 disables this container protection.

A message-driven bean's `onMessage` method can use the `javax.jms.Message` `getJMSRedelivered` method to check whether a received message is a redelivered message.

Note – The `cmt-max-runtime-exceptions` property might be deprecated in the future.

Sample Message-Driven Bean XML Files

This section includes the following sample files:

- "Sample ejb-jar.xml File" on page 160
- "Sample sun-ejb-jar.xml File" on page 161

For general information on the `sun-ejb-jar.xml` file, see "The sun-ejb-jar.xml File" on page 288.

Sample ejb-jar.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
<ejb-jar>
<enterprise-beans>
    <message-driven>
```

```
        <ejb-name>MessageBean</ejb-name>
        <ejb-class>samples.mdb.ejb.MessageBean</ejb-class>
        <transaction-type>Container</transaction-type>
        <message-driven-destination>
            <destination-type>javax.jms.Queue</destination-type>
        </message-driven-destination>
        <resource-ref>
            <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
            <res-type>javax.jms.QueueConnectionFactory</res-type>
            <res-auth>Container</res-auth>
        </resource-ref>
    </message-driven>
</enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>MessageBean</ejb-name>
                <method-intf>Bean</method-intf>
                <method-name>onMessage</method-name>
                <method-params>
                    <method-param>javax.jms.Message</method-param>
                </method-params>
            </method>
        <trans-attribute>NotSupported</trans-attribute>
    </container-transaction>
</assembly-descriptor
</ejb-jar>
```

Sample sun-ejb-jar.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Application
Server 8.1 EJB 2.1//EN'
'http://www.sun.com/software/appserver/dtds/sun-ejb-jar_2_1-1.dtd'>
<sun-ejb-jar>
<enterprise-beans>
    <ejb>
        <ejb-name>MessageBean</ejb-name>
        <jndi-name>jms/sample/Queue</jndi-name>
        <resource-ref>
            <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
            <jndi-name>jms/sample/QueueConnectionFactory</jndi-name>
            <default-resource-principal>
                <name>guest</name>
                <password>guest</password>
            </default-resource-principal>
        </resource-ref>
        <mdb-connection-factory>
```

```
            <jndi-name>jms/sample/QueueConnectionFactory</jndi-name>
            <default-resource-principal>
                <name>guest</name>
                <password>guest</password>
            </default-resource-principal>
        </mdb-connection-factory>
    </ejb>
</enterprise-beans>
</sun-ejb-jar>
```

Handling Transactions with Enterprise Beans

This section describes the transaction support built into the Enterprise JavaBeans programming model for the Application Server.

As a developer, you can write an application that updates data in multiple databases distributed across multiple sites. The site might use EJB servers from different vendors. This section provides overview information on the following topics:

Flat Transactions

The Enterprise JavaBeans Specification, v2.1 requires support for flat (as opposed to nested) transactions. In a flat transaction, each transaction is decoupled from and independent of other transactions in the system. Another transaction cannot start in the same thread until the current transaction ends.

Flat transactions are the most prevalent model and are supported by most commercial database systems. Although nested transactions offer a finer granularity of control over transactions, they are supported by far fewer commercial database systems.

Global and Local Transactions

Understanding the distinction between global and local transactions is crucial in understanding the Application Server support for transactions. See "Transaction Scope" on page 236.

Both local and global transactions are demarcated using the `javax.transaction.UserTransaction` interface, which the client must use. Local transactions bypass the transaction manager and are faster. For more information, see "Naming Environment for J2EE Application Components" on page 240.

Commit Options

The EJB protocol is designed to give the container the flexibility to select the disposition of the instance state at the time a transaction is committed. This allows the container to best manage caching an entity object's state and associating an entity object identity with the EJB instances.

There are three commit-time options:

- **Option A:** The container caches a ready instance between transactions. The container ensures that the instance has exclusive access to the state of the object in persistent storage.

  In this case, the container does *not* have to synchronize the instance's state from the persistent storage at the beginning of the next transaction.

  Note – Commit option A is not supported for this Application Server release.

- **Option B:** The container caches a ready instance between transactions, but the container does *not* ensure that the instance has exclusive access to the state of the object in persistent storage. This is the default.

  In this case, the container must synchronize the instance's state by invoking `ejbLoad` from persistent storage at the beginning of the next transaction.

- **Option C:** The container does *not* cache a ready instance between transactions, but instead returns the instance to the pool of available instances after a transaction has completed.

  The life cycle for every business method invocation under commit option C looks like this:

  ```
  ejbActivate → ejbLoad → business method → ejbStore → ejbPassivate
  ```

  If there is more than one transactional client concurrently accessing the same entity `EJBObject`, the first client gets the ready instance and subsequent concurrent clients get new instances from the pool.

The Application Server deployment descriptor has an element, `commit-option`, that specifies the commit option to be used. Based on the specified commit option, the appropriate handler is instantiated.

Administration and Monitoring

An administrator can control a number of domain-level Transaction Service settings. For details, see "Configuring the Transaction Service" on page 237.

The Transaction Timeout setting can be overridden by a bean. See "Bean-Level Container-Managed Transaction Timeouts" on page 150.

In addition, the administrator can monitor transactions using statistics from the transaction manager that provide information on such activities as the number of transactions completed, rolled back, or recovered since server startup, and transactions presently being processed.

For information on administering and monitoring transactions, see the Application Server Administration Console online help and the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

◆ ◆ ◆ CHAPTER 7

Using Container-Managed Persistence for Entity Beans

This section contains information on how container-managed persistence (CMP) works in the Sun Java System Application Server in the following topics:

Extensive information on CMP is contained in chapters 10, 11, and 14 of the Enterprise JavaBeans Specification, v2.1.

Sun Java System Application Server Support

Application Server support for CMP includes:

- Full support for the J2EE v 1.4 specification's CMP model.
  - Support for commit options B and C for transactions, as defined in the Enterprise JavaBeans Specification, v2.1. See "Commit Options" on page 163.
  - The primary key class must be a subclass of `java.lang.Object`. This ensures portability, and is noted because some vendors allow primitive types (such as `int`) to be used as the primary key class.
- The Application Server CMP implementation, which provides:
  - An Object/Relational (O/R) mapping tool that creates XML deployment descriptors for EJB JAR files that contain beans that use CMP
  - Support for compound (multi-column) primary keys

- Support for sophisticated custom finder methods
- Standards-based query language (EJB QL)
- CMP runtime support. See "Configuring the CMP Resource" on page 177.

- Application Server performance-related features, including:
  - Version column consistency checking
  - Relationship prefetching
  - Read-Only Beans

  For details, see "Performance-Related Features" on page 182.

Container-Managed Persistence Mapping

Implementation for entity beans that use CMP is mostly a matter of mapping CMP fields and CMR fields (relationships) to the database. This section addresses the following topics:

Mapping Capabilities

Mapping refers to the ability to tie an object-based model to a relational model of data, usually the schema of a relational database. The CMP implementation provides the ability to tie a set of interrelated beans containing data and associated behaviors to the schema. This object representation of the database becomes part of the Java application. You can also customize this mapping to optimize these beans for the particular needs of an application. The result is a single data model through which both persistent database information and regular transient program data are accessed.

The mapping capabilities provided by the Application Server include:

- Mapping a CMP bean to one or more tables
- Mapping CMP fields to one or more columns
- Mapping CMP fields to different column types
- Mapping tables with compound primary keys
- Mapping tables with unknown primary keys
- Mapping CMP relationships to foreign keys
- Mapping tables with overlapping primary and foreign keys

The Mapping Deployment Descriptor File

Each module with CMP beans must have the following files:

- `ejb-jar.xml`: The J2EE standard file for assembling enterprise beans. For a detailed description, see the Enterprise JavaBeans Specification, v2.1.
- `sun-ejb-jar.xml`: The Application Server standard file for assembling enterprise beans. For a detailed description, see "The sun-ejb-jar.xml File" on page 288.
- `sun-cmp-mappings.xml`: The *mapping deployment descriptor file*, which describes the mapping of CMP beans to tables in a database. For a detailed description, see "The sun-cmp-mappings.xml File" on page 293.

The `sun-cmp-mappings.xml` file can be automatically generated and does not have to exist prior to deployment. For details, see "Generation Options" on page 173.

The `sun-cmp-mappings.xml` file maps CMP fields and CMR fields (relationships) to the database. A primary table must be selected for each CMP bean, and optionally, multiple secondary tables. CMP fields are mapped to columns in either the primary or secondary table(s). CMR fields are mapped to pairs of column lists (normally, column lists are the lists of columns associated with primary and foreign keys).

Note – Table names in databases can be case-sensitive. Make sure that the table names in the `sun-cmp-mappings.xml` file match the names in the database.

Relationships should always be mapped to the primary key field(s) of the related table.

The `sun-cmp-mappings.xml` file conforms to the `sun-cmp-mapping_1_2.dtd` file and is packaged with the user-defined bean classes in the EJB JAR file under the `META-INF` directory.

The Application Server or the deploytool creates the mappings in the `sun-cmp-mappings.xml` file automatically during deployment if the file is not present. For information on how to use the deploytool for mapping, see the "Create Database Mapping" topic in the deploytool's online help.

To map the fields and relationships of your entity beans manually, edit the `sun-cmp-mappings.xml` deployment descriptor. Only do this if you are proficient in editing XML.

The mapping information is developed in conjunction with the database schema (`.dbschema`) file, which can be automatically captured when you deploy the bean (see "Automatic Database Schema Capture" on page 176). You can manually generate the schema using the `capture-schema` utility ("Using the capture-schema Utility" on page 176.

Mapping Considerations

This section addresses the following topics:

The data types used in automatic schema generation are also suggested for manual mapping. These data types are described in "Supported Data Types" on page 171.

Join Tables and Relationships

Use of join tables in the database schema is supported for all types of relationships, not just many-to-many relationships. For general information about relationships, see section 10.3.7 of the Enterprise JavaBeans Specification, v2.1.

Automatic Primary Key Generation

The Application Server supports automatic primary key generation for EJB 1.1, 2.0, and 2.1 CMP beans. To specify automatic primary key generation, give the `prim-key-class` element in the `ejb-jar-xml` file the value `java.lang.Object`. CMP beans with automatically generated primary keys can participate in relationships with other CMP beans. The Application Server does not support database-generated primary key values.

If the database schema is created during deployment, the Application Server creates the schema with the primary key column, then generates unique values for the primary key column at runtime.

If the database schema is not created during deployment, the primary key column in the mapped table must be of type `NUMERIC` with a precision of 19 or more, and must not be mapped to any CMP field. The Application Server generates unique values for the primary key column at runtime.

Fixed Length CHAR Primary Keys

If an existing database table has a primary key column in which the values vary in length, but the type is `CHAR` instead of `VARCHAR`, the Application Server automatically trims any extra spaces when retrieving primary key values. It is not a good practice to use a fixed length `CHAR` column as a primary key. Use this feature with schemas that cannot be changed, such as a schema inherited from a legacy application.

Managed Fields

A managed field is a CMP or CMR field that is mapped to the same database column as another CMP or CMR field. CMP fields mapped to the same column and CMR fields mapped to exactly the same column lists always have the same value in memory. For CMR fields that share only a subset of their mapped columns, changes to the columns affect the relationship fields in memory differently. Basically, the Application Server always tries to keep the state of the objects in memory synchronized with the database.

A managed field can have any "fetched-with" on page 335 subelement except `<default/>`.

BLOB Support

Binary Large Object (BLOB) is a data type used to store values that do not correspond to other types such as numbers, strings, or dates. Java fields whose types implement `java.io.Serializable` or are represented as `byte[]` can be stored as BLOBs.

If a CMP field is defined as `Serializable`, it is serialized into a `byte[]` before being stored in the database. Similarly, the value fetched from the database is deserialized. However, if a CMP field is defined as `byte[]`, it is stored directly instead of being serialized and deserialized when stored and fetched, respectively.

To enable BLOB support in the Application Server environment, define a CMP field of type `byte[]` or a user-defined type that implements the `java.io.Serializable` interface. If you map the CMP bean to an existing database schema, map the field to a column of type BLOB.

To use BLOB or CLOB data types larger than 4 KB for CMP using the Inet Oraxo JDBC Driver for Oracle 8.1.7 and 9.x Databases, you must set the `streamstolob` property value to `true`.

For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

For automatic mapping, you might need to change the default BLOB column length for the generated schema using the `schema-generator-properties` element in `sun-ejb-jar.xml`. See your database vendor documentation to determine whether you need to specify the length. For example:

```
<schema-generator-properties>
   <property>
      <name>Employee.voiceGreeting.jdbc-type</name>
      <value>BLOB</value>
   </property>
   <property>
      <name>Employee.voiceGreeting.jdbc-maximum-length</name>
      <value>10240</value>
   </property>
   ...
</schema-generator-properties>
```

CLOB Support

Character Large Object (CLOB) is a data type used to store and retrieve very long text fields. CLOBs translate into long strings.

To enable CLOB support in the Application Server environment, define a CMP field of type `java.lang.String`. If you map the CMP bean to an existing database schema, map the field to a column of type CLOB.

To use BLOB or CLOB data types larger than 4 KB for CMP using the Inet Oraxo JDBC Driver for Oracle 8.1.7 and 9.x Databases, you must set the `streamstolob` property value to `true`.

For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

For automatic mapping, you might need to change the default CLOB column length for the generated schema using the `schema-generator-properties` element in `sun-ejb-jar.xml`. See your database vendor documentation to determine whether you need to specify the length. For example:

```
<schema-generator-properties>
   <property>
      <name>Employee.resume.jdbc-type</name>
      <value>CLOB</value>
   </property>
   <property>
      <name>Employee.resume.jdbc-maximum-length</name>
      <value>10240</value>
   </property>
   ...
</schema-generator-properties>
```

Automatic Schema Generation

The automatic schema generation feature provided in the Application Server defines database tables based on the fields in entity beans and the relationships between the fields. This insulates developers from many of the database related aspects of development, allowing them to focus on entity bean development. The resulting schema is usable as-is or can be given to a database administrator for tuning with respect to performance, security, and so on.

This section addresses the following topics:

Supported Data Types

CMP supports a set of JDBC data types that are used in mapping Java data fields to SQL types. Supported JDBC data types are as follows: BIGINT, BIT, BLOB, CHAR, CLOB, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, VARCHAR.

The following table contains the mappings of Java types to JDBC types when automatic mapping is used.

TABLE 7–1 Java Type to JDBC Type Mappings

Java Type	JDBC Type	Nullability
`boolean`	`BIT`	No
`java.lang.Boolean`	`BIT`	Yes
`byte`	`TINYINT`	No
`java.lang.Byte`	`TINYINT`	Yes
`double`	`DOUBLE`	No
`java.lang.Double`	`DOUBLE`	Yes
`float`	`REAL`	No
`java.lang.Float`	`REAL`	Yes
`int`	`INTEGER`	No
`java.lang.Integer`	`INTEGER`	Yes
`long`	`BIGINT`	No
`java.lang.Long`	`BIGINT`	Yes
`short`	`SMALLINT`	No
`java.lang.Short`	`SMALLINT`	Yes
`java.math.BigDecimal`	`DECIMAL`	Yes
`java.math.BigInteger`	`DECIMAL`	Yes
`char`	`CHAR`	No
`java.lang.Character`	`CHAR`	Yes
`java.lang.String`	`VARCHAR` or `CLOB`	Yes
`Serializable`	`BLOB`	Yes

TABLE 7–1 Java Type to JDBC Type Mappings *(Continued)*

Java Type	JDBC Type	Nullability
`byte[]`	`BLOB`	Yes
`java.util.Date`	`DATE` (Oracle only) `TIMESTAMP` (all other databases)	Yes
`java.sql.Date`	`DATE`	Yes
`java.sql.Time`	`TIME`	Yes
`java.sql.Timestamp`	`TIMESTAMP`	Yes

Note – Java types assigned to CMP fields must be restricted to Java primitive types, Java `Serializable` types, `java.util.Date`, `java.sql.Date`, `java.sql.Time`, or `java.sql.Timestamp`. An entity bean local interface type (or a collection of such) can be the type of a CMR field.

The following table contains the mappings of JDBC types to database vendor-specific types when automatic mapping is used. For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

TABLE 7–2 Mappings of JDBC Types to Database Vendor Specific Types

JDBC Type	Derby	Oracle	DB2	Sybase ASE 12.5	MS-SQL Server
`BIT`	`SMALLINT`	`SMALLINT`	`SMALLINT`	`TINYINT`	`BIT`
`TINYINT`	`SMALLINT`	`SMALLINT`	`SMALLINT`	`TINYINT`	`TINYINT`
`SMALLINT`	`SMALLINT`	`SMALLINT`	`SMALLINT`	`SMALLINT`	`SMALLINT`
`INTEGER`	`INTEGER`	`INTEGER`	`INTEGER`	`INTEGER`	`INTEGER`
`BIGINT`	`BIGINT`	`NUMBER`	`BIGINT`	`NUMERIC`	`NUMERIC`
`REAL`	`REAL`	`REAL`	`FLOAT`	`FLOAT`	`REAL`
`DOUBLE`	`DOUBLE PRECISION`	`DOUBLE PRECISION`	`DOUBLE`	`DOUBLE PRECISION`	`FLOAT`
`DECIMAL(p,s)`	`DECIMAL(p,s)`	`NUMBER(p,s)`	`DECIMAL(p,s)`	`DECIMAL(p,s)`	`DECIMAL(p,s)`
`VARCHAR`	`VARCHAR`	`VARCHAR2`	`VARCHAR`	`VARCHAR`	`VARCHAR`
`DATE`	`DATE`	`DATE`	`DATE`	`DATETIME`	`DATETIME`

TABLE 7–2 Mappings of JDBC Types to Database Vendor Specific Types *(Continued)*

JDBC Type	Derby	Oracle	DB2	Sybase ASE 12.5	MS-SQL Server
TIME	TIME	DATE	TIME	DATETIME	DATETIME
TIMESTAMP	TIMESTAMP	TIMESTAMP(9)	TIMESTAMP	DATETIME	DATETIME
BLOB	BLOB	BLOB	BLOB	IMAGE	IMAGE
CLOB	CLOB	CLOB	CLOB	TEXT	NTEXT

Generation Options

Deployment descriptor elements or `asadmin` command line options can control automatic schema generation by:

- Creating tables during deployment
- Dropping tables during undeployment
- Dropping and creating tables during redeployment
- Specifying the database vendor
- Specifying that table names are unique
- Specifying type mappings for individual CMP fields

Note – Before using these options, make sure you have a properly configured CMP resource. See "Configuring the CMP Resource" on page 177.

You can also use the deploytool to perform automatic mapping. For more information about using the deploytool, see the "Create Database Mapping" topic in the deploytool's online help.

For a read-only bean, do not create the database schema during deployment. Instead, work with your database administrator to populate the data into the tables. See "Using Read-Only Beans" on page 154.

Automatic schema generation is not supported for beans with version column consistency checking. Instead, work with your database administrator to create the schema and add the required triggers. See "Version Column Consistency Checking" on page 182.

The following optional data subelements of the `cmp-resource` element in the `sun-ejb-jar.xml` file control the automatic creation of database tables at deployment. For more information about the `cmp-resource` element, see "cmp-resource" on page 316 and "Configuring the CMP Resource" on page 177.

TABLE 7–3 `sun-ejb-jar.xml` Generation Elements

Element	Default	Description
"create-tables-at-deploy" on page 323	`false`	If `true`, causes database tables to be created for beans that are automatically mapped by the EJB container. If `false`, does not create tables.
"drop-tables-at-undeploy" on page 326	`false`	If `true`, causes database tables that were automatically created when the bean(s) were last deployed to be dropped when the bean(s) are undeployed. If `false`, does not drop tables.
"database-vendor-name" on page 324	none	Specifies the name of the database vendor for which tables are created. Allowed values are `db2`, `mssql`, `oracle`, `derby`, and `sybase`, case-insensitive. If no value is specified, a connection is made to the resource specified by the `jndi-name` subelement of the `cmp-resource` element in the `sun-ejb-jar.xml` file, and the database vendor name is read. If the connection cannot be established, or if the value is not recognized, SQL-92 compliance is presumed.
"schema-generator-properties" on page 383	none	Specifies field-specific column attributes in `property` subelements. Each property name is of the following format: *bean-name*.*field-name*.*attribute* For example: `Employee.firstName.jdbc-type` Column attributes are described in Table A–95. Also allows you to set the `use-unique-table-names` property. If `true`, this property specifies that generated table names are unique within each application server domain. The default is `false`. For further information and an example, see "schema-generator-properties" on page 383.

The following options of the `asadmin deploy` or `asadmin deploydir` command control the automatic creation of database tables at deployment:

TABLE 7–4 `asadmin deploy` and `asadmin deploydir` Generation Options

Option	Default	Description
`--createtables`	none	If `true`, causes database tables to be created for beans that need them. If `false`, does not create tables. If not specified, the value of the `create-tables-at-deploy` attribute in `sun-ejb-jar.xml` is used.

TABLE 7–4 `asadmin deploy` and `asadmin deploydir` Generation Options *(Continued)*

Option	Default	Description
`--dropandcreatetables`	none	If `true`, and if tables were automatically created when this application was last deployed, tables from the earlier deployment are dropped and fresh ones are created. If `true`, and if tables were *not* automatically created when this application was last deployed, no attempt is made to drop any tables. If tables with the same names as those that would have been automatically created are found, the deployment proceeds, but a warning indicates that tables could not be created. If `false`, settings of `create-tables-at-deploy` or `drop-tables-at-undeploy` in the `sun-ejb-jar.xml` file are overridden.
`--uniquetablenames`	none	If `true`, specifies that table names are unique within each application server domain. If not specified, the value of the `use-unique-table-names` property in `sun-ejb-jar.xml` is used.
`--dbvendorname`	none	Specifies the name of the database vendor for which tables are created. Allowed values are `db2`, `mssql`, `oracle`, `derby`, and `sybase`, case-insensitive. If not specified, the value of the `database-vendor-name` attribute in `sun-ejb-jar.xml` is used. If no value is specified, a connection is made to the resource specified by the `jndi-name` subelement of the `cmp-resource` element in the `sun-ejb-jar.xml` file, and the database vendor name is read. If the connection cannot be established, or if the value is not recognized, SQL-92 compliance is presumed.

If one or more of the beans in the module are manually mapped and you use any of the `asadmin deploy` or `asadmin deploydir` options, the deployment is not harmed in any way, but the options have no effect, and a warning is written to the server log.

If the deploytool mapped one or more of the beans, the `--uniquetablenames` option of `asadmin deploy` or `asadmin deploydir` has no effect. The uniqueness of the table names was established when deploytool created the mapping.

The following options of the `asadmin undeploy` command control the automatic removal of database tables at undeployment:

TABLE 7–5 `asadmin undeploy` Generation Options

Option	Default	Description
`--droptables`	none	If `true`, causes database tables that were automatically created when the bean(s) were last deployed to be dropped when the bean(s) are undeployed. If `false`, does not drop tables. If not specified, the value of the `drop-tables-at-undeploy` attribute in `sun-ejb-jar.xml` is used.

For more information about the `asadmin deploy`, `asadmin deploydir`, and `asadmin undeploy` commands, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

When command line and `sun-ejb-jar.xml` options are both specified, the `asadmin` options take precedence.

Schema Capture

This section addresses the following topics:

Automatic Database Schema Capture

You can configure a CMP bean in Application Server to automatically capture the database metadata and save it in a `.dbschema` file during deployment. If the `sun-cmp-mappings.xml` file contains an empty `<schema/>` entry, the `cmp-resource` entry in the `sun-ejb-jar.xml` file is used to get a connection to the database, and automatic generation of the schema is performed.

Note – Before capturing the database schema automatically, make sure you have a properly configured CMP resource. See "Configuring the CMP Resource" on page 177.

Using the capture-schema Utility

You can use the `capture-schema` command to manually generate the database metadata (`.dbschema`) file. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

The `capture-schema` utility does *not* modify the schema in any way. Its only purpose is to provide the persistence engine with information about the structure of the database (the schema).

Keep the following in mind when using the `capture-schema` command:

- The name of a `.dbschema` file must be unique across all deployed modules in a domain.
- If more than one schema is accessible for the schema user, more than one table with the same name might be captured if the `-schemaname` parameter of `capture-schema` is not set.
- The schema name must be upper case.
- Table names in databases are case-sensitive. Make sure that the table name matches the name in the database.
- An Oracle database user running the `capture-schema` command needs ANALYZE ANY TABLE privileges if that user does not own the schema. These privileges are granted to the user by the database administrator.

Configuring the CMP Resource

An EJB module that contains CMP beans requires the JNDI name of a JDBC resource or Persistence Manager resource in the `jndi-name` subelement of the "cmp-resource" on page 316 element in the `sun-ejb-jar.xml` file. If the JNDI name refers to a JDBC Resource, set `PersistenceManagerFactory` properties as properties of the `cmp-resource` element in the `sun-ejb-jar.xml` file.

In the Administration Console, open the Resources component, then select JDBC or Persistence Managers. Refer to the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* for information on creating a new CMP resource.

For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

For example, if the JDBC resource has the JNDI name `jdbc/MyDatabase`, set the CMP resource in the `sun-ejb-jar.xml` file as follows:

```
<cmp-resource>
   <jndi-name>jdbc/MyDatabase</jndi-name>
</cmp-resource>
```

For another example, if the Persistence Manager has the JNDI name `jdo/MyDatabase`, set the CMP resource in the `sun-ejb-jar.xml` file as follows:

```
<cmp-resource>
   <jndi-name>jdo/MyDatabase</jndi-name>
</cmp-resource>
```

Configuring Queries for 1.1 Finders

This section contains the following topics:

About JDOQL Queries

The Enterprise JavaBeans Specification, v1.1 spec does not specify the format of the finder method description. The Application Server uses an extension of Java Data Objects Query Language (JDOQL) queries to implement finder and selector methods. (For EJB 2.0 and later, the container automatically maps an EJB QL query to JDOQL.) You can specify the following elements of the underlying JDOQL query:

- **Filter expression** - A Java-like expression that specifies a condition that each object returned by the query must satisfy. Corresponds to the WHERE clause in EJB QL.
- **Query parameter declaration** - Specifies the name and the type of one or more query input parameters. Follows the syntax for formal parameters in the Java language.
- **Query variable declaration** - Specifies the name and type of one or more query variables. Follows the syntax for local variables in the Java language. A query filter might use query variables to implement joins.
- **Query ordering declaration** - Specifies the ordering expression of the query. Corresponds to the ORDER BY clause of EJB QL.

The Application Server specific deployment descriptor (`sun-ejb-jar.xml`) provides the following elements to store the EJB 1.1 finder method settings:

```
query-filterquery-paramsquery-variablesquery-ordering
```

The bean developer uses these elements to construct a query. When the finder method that uses these elements executes, the values of these elements are used to execute a query in the database. The objects from the JDOQL query result set are converted into primary key instances to be returned by the EJB 1.1 `ejbFind` method.

The JDO specification (see JSR 12) provides a comprehensive description of JDOQL. The following information summarizes the elements used to define EJB 1.1 finders.

Query Filter Expression

The filter expression is a String containing a boolean expression evaluated for each instance of the candidate class. If the filter is not specified, it defaults to true. Rules for constructing valid expressions follow the Java language, with the following differences:

- Equality and ordering comparisons between primitives and instances of wrapper classes are valid.
- Equality and ordering comparisons of Date fields and Date parameters are valid.
- Equality and ordering comparisons of String fields and String parameters are valid.
- White space (non-printing characters space, tab, carriage return, and line feed) is a separator and is otherwise ignored.
- The following assignment operators are not supported:
  - =, +=, etc.
  - pre- and post-increment
  - pre- and post-decrement
- Methods, including object construction, are not supported, except for:

```
Collection.contains(Object o)
Collection.isEmpty()
String.startsWith(String s)
String.endsWith(String e)
```

  In addition, the Application Server supports the following nonstandard JDOQL methods:

```
String.like(String pattern)
String.like(String pattern, char escape)
String.substring(int start, int length)
String.indexOf(String str)
String.indexOf(String str, int start)
String.length()
Math.abs(numeric n)
Math.sqrt(double d)
```

- Navigation through a null-valued field, which throws a `NullPointerException`, is treated as if the sub-expression returned `false`.

Note – Comparisons between floating point values are by nature inexact. Therefore, equality comparisons (== and !=) with floating point values should be used with caution. Identifiers in the expression are considered to be in the name space of the candidate class, with the addition of declared parameters and variables. As in the Java language, `this` is a reserved word, and refers to the current instance being evaluated.

The following expressions are supported:

- Operators applied to all types where they are defined in the Java language:
  - relational operators (==, !=, >, <, >=, <=)
  - boolean operators (&, &&, |, ||, ~, !)
  - arithmetic operators (+, -, *, /)

  String concatenation is supported only for String + String.
- Parentheses to explicitly mark operator precedence
- Cast operator
- Promotion of numeric operands for comparisons and arithmetic operations. The rules for promotion follow the Java rules (see the numeric promotions of the Java language specification) extended by BigDecimal, BigInteger, and numeric wrapper classes.

Query Parameters

The parameter declaration is a String containing one or more parameter type declarations separated by commas. This follows the Java syntax for method signatures.

Query Variables

The type declarations follow the Java syntax for local variable declarations.

JDOQL Examples

This section provides a few query examples.

Example 1

The following query returns all players called Michael. It defines a filter that compares the name field with a string literal:

```
name == "Michael"
```

The `finder` element of the `sun-ejb-jar.xml` file looks like this:

```
<finder>
   <method-name>findPlayerByName</method-name>
   <query-filter>name == "Michael"</query-filter>
</finder>
```

Example 2

This query returns all products in a specified price range. It defines two query parameters which are the lower and upper bound for the price: double low, double high. The filter compares the query parameters with the price field:

```
low < price && price < high
```

Query ordering is set to `price ascending`.

The `finder` element of the `sun-ejb-jar.xml` file looks like this:

```
<finder>
   <method-name>findInRange</method-name>
   <query-params>double low, double high</query-params>
   <query-filter>low &lt; price &amp;&amp; price &lt high</query-filter>
   <query-ordering>price ascending</query-ordering>
</finder>
```

Example 3

This query returns all players having a higher salary than the player with the specified name. It defines a query parameter for the name `java.lang.String name`. Furthermore, it defines a variable to which the player's salary is compared. It has the type of the persistence capable class that corresponds to the bean:

```
    mypackage.PlayerEJB_170160966_JDOState player
```

The filter compares the salary of the current player denoted by the `this` keyword with the salary of the player with the specified name:

```
    (this.salary > player.salary) && (player.name == name)
```

The `finder` element of the `sun-ejb-jar.xml` file looks like this:

```
<finder>
   <method-name>findByHigherSalary</method-name>
   <query-params>java.lang.String name</query-params>
   <query-filter>
      (this.salary &gt; player.salary) &amp;&amp; (player.name == name)
   </query-filter>
   <query-variables>
      mypackage.PlayerEJB_170160966_JDOState player
   </query-variables>
</finder>
```

Performance-Related Features

The Application Server provides the following features to enhance performance or allow more fine-grained data checking. These features are supported only for entity beans with container managed persistence.

Note – Use of any of these features results in a non-portable application.

Version Column Consistency Checking

The version consistency feature saves the bean state at first transactional access and caches it between transactions. The state is copied from the cache instead of being read from the database. The bean state is verified by primary key and version column values at flush for custom queries (for dirty instances only) and at commit (for clean and dirty instances).

▼ To use version consistency

1. **Create the version column in the primary table.**
2. **Give the version column a numeric data type.**
3. **Provide appropriate update triggers on the version column.**

   These triggers must increment the version column on each update of the specified row.
4. **Specify the version column.**

   This is specified in the "check-version-of-accessed-instances" on page 311 subelement of the "consistency" on page 320 element in the `sun-cmp-mappings.xml` file.
5. **Map the CMP bean to an existing schema.**

   Automatic schema generation is not supported for beans with version column consistency checking. Instead, work with your database administrator to create the schema and add the required triggers.

Relationship Prefetching

In many cases when an entity bean's state is fetched from the database, its relationship fields are always accessed in the same transaction. Relationship prefetching saves database round trips by fetching data for an entity bean and those beans referenced by its CMR fields in a single database round trip.

To enable relationship prefetching for a CMR field, use the "default" on page 324 subelement of the "fetched-with" on page 335 element in the `sun-cmp-mappings.xml` file. By default, these CMR fields are prefetched whenever `findByPrimaryKey` or a custom finder is executed for the entity, or when the entity is navigated to from a relationship. (Recursive prefetching is not supported, because it does not usually enhance performance.) To disable prefetching for specific custom finders, use the "prefetch-disabled" on page 367 element in the `sun-ejb-jar.xml` file.

Read-Only Beans

Another feature that the Application Server provides is the *read-only bean*, an entity bean that is never modified by an EJB client. Read-only beans avoid database updates completely.

A read-only bean can be used to cache a database entry that is frequently accessed but rarely updated (externally by other beans). When the data that is cached by a read-only bean is updated by another bean, the read-only bean can be notified to refresh its cached data.

The Application Server provides a number of ways by which a read-only bean's state can be refreshed. By setting the `refresh-period-in-seconds` element in the `sun-ejb-jar.xml` file and the `trans-attribute` element in the `ejb-jar.xml` file, it is easy to configure a read-only bean that is (a) always refreshed, (b) periodically refreshed, (c) never refreshed, or (d) programmatically refreshed.

Access to CMR fields of read-only beans is not supported. Deployment will succeed, but an exception will be thrown at runtime if a get or set method is invoked.

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently. For further information and usage guidelines, see "Using Read-Only Beans" on page 154.

Restrictions and Optimizations

This section discusses restrictions and performance optimizations that pertain to using CMP entity beans.

Eager Loading of Field State

By default, the EJB container loads the state for all CMP fields (excluding relationship, BLOB, and CLOB fields) before invoking the `ejbLoad` method of the abstract bean. This approach might not be optimal for entity objects with large state if most business methods require access to only parts of the state. If this is an issue, use the "fetched-with" on page 335 element in `sun-cmp-mappings.xml` for fields that are used infrequently.

Restrictions on Remote Interfaces

The following restrictions apply to the remote interface of an entity bean that uses CMP:

- Do not expose the `get` and `set` methods for CMR fields or the persistence collection classes that are used in container-managed relationships through the remote interface of the bean.

  However, you are free to expose the `get` and `set` methods that correspond to the CMP fields of the entity bean through the bean's remote interface.
- Do not expose the container-managed collection classes that are used for relationships through the remote interface of the bean.
- Do not expose local interface types or local home interface types through the remote interface or remote home interface of the bean.

Dependent value classes can be exposed in the remote interface or remote home interface, and can be included in the client EJB JAR file.

Sybase Finder Limitation

If a finder method with an input greater than 255 characters is executed and the primary key column is mapped to a VARCHAR column, Sybase attempts to convert type VARCHAR to type TEXT and generates the following error:

```
com.sybase.jdbc2.jdbc.SybSQLException: Implicit conversion from datatype
'TEXT' to 'VARCHAR' is not allowed.  Use the CONVERT function to run this
query.
```

To avoid this error, make sure the finder method input is less than 255 characters.

Date and Time Fields as CMP Field Types

If a CMP field type is a Java date or time type (`java.util.Date`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`), make sure that the field value exactly matches the value in the database.

For example, the following code uses a `java.sql.Date` type as a primary key field:

```
java.sql.Date myDate = new java.sql.Date(System.currentTimeMillis())
beanHome.create(myDate, ...);
```

For some databases, this code results in only the year, month, and date portion of the field value being stored in the database. Later on if the client tries to find this bean by primary key as follows:

```
myBean = beanHome.findByPrimaryKey(myDate);
```

the bean is not found in the database because the value does not match the one that is stored in the database.

Similar problems can happen if the database truncates the timestamp value while storing it, or if a custom query has a date or time value comparison in its WHERE clause.

For automatic mapping to an Oracle database, fields of type `java.util.Date`, `java.sql.Date`, and `java.sql.Time` are mapped to Oracle's DATE data type. Fields of type `java.sql.Timestamp` are mapped to Oracle's `TIMESTAMP(9)` data type.

No Support for lock-when-loaded on Sybase and DB2

The "lock-when-loaded" on page 349 consistency level is implemented by placing update locks on the data corresponding to a bean when the data is loaded from the database. There is no suitable mechanism available on Sybase and DB2 databases to implement this feature. Therefore, the lock-when-loaded "consistency" on page 320 level is not supported on Sybase and DB2 databases.

Set RECURSIVE_TRIGGERS to false on MSSQL

For version consistency triggers on MSSQL, the property `RECURSIVE_TRIGGERS` must be set to `false`, which is the default. If set to `true`, triggers throw a `java.sql.SQLException`.

Set this property as follows:

```
EXEC sp_dboption 'database-name', 'recursive triggers', 'FALSE'
go
```

You can test this property as follows:

```
SELECT DATABASEPROPERTYEX('database-name', 'IsRecursiveTriggersEnabled')
go
```

MySQL Database Restrictions

The following restrictions apply when you use a MySQL database with the Application Server for CMP.

- MySQL treats `int1` and `int2` as reserved words. If you want to define `int1` and `int2` as fields in your table, use ‘`int1`‘ and ‘`int2`‘ field names in your SQL file.
- When `VARCHAR` fields get truncated, a warning is displayed instead of an error. To get an error message, start the MySQL database in strict SQL mode.
- The order of fields in a foreign key index must match the order in the explicitly created index on the primary table.
- The `CREATE TABLE` syntax in the SQL file must end with the following line:

  ```
  )  Engine=InnoDB;
  ```

  `InnoDB` provides MySQL with a transaction-safe (ACID compliant) storage engine having commit, rollback, and crash recovery capabilities.
- For a `FLOAT` type field, the correct precision must be defined. By default, MySQL uses four bytes to store a `FLOAT` type that does not have an explicit precision definition. For example, this causes a number such as 12345.67890123 to be rounded off to 12345.7 during an `INSERT`. To prevent this, specify `FLOAT(10,2)` in the DDL file, which forces the database to use an eight-byte double-precision column. For more information, see `http://dev.mysql.com/doc/mysql/en/numeric-types.html`.
- To use `||` as the string concatenation symbol, start the MySQL server with the `--sql-mode="PIPES_AS_CONCAT"` option. For more information, see `http://dev.mysql.com/doc/refman/5.0/en/server-sql-mode.html` and `http://dev.mysql.com/doc/mysql/en/ansi-mode.html`.

- MySQL always starts a new connection when `autoCommit==true` is set. This ensures that each SQL statement forms a single transaction on its own. If you try to rollback or commit an SQL statement, you get an error message:

```
javax.transaction.SystemException: java.sql.SQLException:
Can't call rollback when autocommit=true

javax.transaction.SystemException: java.sql.SQLException:
Error open transaction is not closed
```

  To resolve this issue, add `relaxAutoCommit=true` to the JDBC URL. For more information, see `http://forums.mysql.com/read.php?39,31326,31404`.

- Change the trigger create format from the following:

```
CREATE TRIGGER T_UNKNOWNPKVC1
BEFORE UPDATE ON UNKNOWNPKVC1
FOR EACH ROW
        WHEN (NEW.VERSION = OLD.VERSION)
BEGIN
        :NEW.VERSION := :OLD.VERSION + 1;
END;
/
```

  to the following:

```
DELIMITER |
CREATE TRIGGER T_UNKNOWNPKVC1
BEFORE UPDATE ON UNKNOWNPKVC1
FOR EACH ROW
        WHEN (NEW.VERSION = OLD.VERSION)
BEGIN
        :NEW.VERSION := :OLD.VERSION + 1;
END
|
DELIMITER ;
```

  For more information, see `http://dev.mysql.com/doc/mysql/en/create-trigger.html`.

- MySQL does not allow a `DELETE` on a row that contains a reference to itself. Here is an example that illustrates the issue:

```
create table EMPLOYEE (
        empId   int         NOT NULL,
        salary  float(25,2) NULL,
        mgrId   int         NULL,
        PRIMARY KEY (empId),
        FOREIGN KEY (mgrId) REFERENCES EMPLOYEE (empId)
        ) ENGINE=InnoDB;
```

```
        insert into Employee values (1, 1234.34, 1);
        delete from Employee where empId = 1;
```

This example fails with the following error message:

```
ERROR 1217 (23000): Cannot delete or update a parent row:
a foreign key constraint fails
```

To resolve this issue, change the table creation script to the following:

```
create table EMPLOYEE (
        empId   int         NOT NULL,
        salary  float(25,2) NULL,
        mgrId   int         NULL,
        PRIMARY KEY (empId),
        FOREIGN KEY (mgrId) REFERENCES EMPLOYEE (empId)
        ON DELETE SET NULL
        ) ENGINE=InnoDB;

        insert into Employee values (1, 1234.34, 1);
        delete from Employee where empId = 1;
```

This can be done only if the foreign key field is allowed to be null. For more information, see `http://bugs.mysql.com/bug.php?id=12449` and `http://dev.mysql.com/doc/mysql/en/innodb-foreign-key-constraints.html`.

- When an SQL script has foreign key constraints defined, `capture-schema` fails to capture the table information correctly. To work around the problem, remove the constraints and then run `capture-schema`. Here is an example that illustrates the issue:

```
CREATE TABLE ADDRESSBOOKBEANTABLE (ADDRESSBOOKNAME VARCHAR(255)
    NOT NULL PRIMARY KEY,
CONNECTEDUSERS              BLOB NULL,
OWNER                       VARCHAR(256),
FK_FOR_ACCESSPRIVILEGES     VARCHAR(256),
CONSTRAINT FK_ACCESSPRIVILEGE FOREIGN KEY (FK_FOR_ACCESSPRIVILEGES)
    REFERENCES ACCESSPRIVILEGESBEANTABLE (ROOT)
) ENGINE=InnoDB;
```

To resolve this issue, change the table creation script to the following:

```
CREATE TABLE ADDRESSBOOKBEANTABLE (ADDRESSBOOKNAME VARCHAR(255)
    NOT NULL PRIMARY KEY,
CONNECTEDUSERS              BLOB NULL,
OWNER                       VARCHAR(256),
FK_FOR_ACCESSPRIVILEGES     VARCHAR(256)
) ENGINE=InnoDB;
```

◆ ◆ ◆ CHAPTER 8

Developing Java Clients

This chapter describes how to develop, assemble, and deploy J2EE Application Clients in the following sections:

Introducing the Application Client Container

The Application Client Container (ACC) includes a set of Java classes, libraries, and other files that are required for and distributed with Java client programs that execute in their own Java Virtual Machine (JVM). The ACC manages the execution of J2EE application client components, which are used to access a variety of J2EE services (such as JMS resources, EJB components, web services, security, and so on.) from a JVM outside the Sun Java System Application Server.

The ACC communicates with the Application Server using RMI-IIOP protocol and manages the details of RMI-IIOP communication using the client ORB that is bundled with it. Compared to other J2EE containers, the ACC is lightweight.

Security

The ACC is responsible for collecting authentication data such as the username and password and sending the collected data to the Application Server. The Application Server then processes the authentication data using the configured Java™ Authentication and Authorization Service (JAAS) module.

Authentication techniques are provided by the client container, and are not under the control of the application client component. The container integrates with the platform's authentication

system. When you execute a client application, it displays a login window and collects authentication data from the user. It also supports SSL (Secure Socket Layer)/IIOP if configured and when necessary.

Naming

The client container enables the application clients to use the Java Naming and Directory Interface (JNDI) to look up J2EE services (such as JMS resources, EJB components, web services, security, and so on.) and to reference configurable parameters set at the time of deployment.

Developing Clients Using the ACC

This section describes the procedure to develop, assemble, and deploy client applications using the ACC. This section describes the following topics:

For information about Java-based clients that are not packaged using the ACC, see "Developing Clients Without the ACC" on page 196.

▼ To access an EJB component from an application client

1 **In your client code, instantiate the `InitialContext` using the default (no argument) constructor:**

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

It is not necessary to explicitly instantiate a naming context that points to the CosNaming service.

2 **In your client code, look up the home object by specifying the JNDI name of the home object as specified in the `ejb-jar.xml` file.**

For example:

```
Object ref = ctx.lookup("java:comp/env/ejb-ref-name");
BeanAHome = (BeanAHome)PortableRemoteObject.narrow(ref,BeanAHome.class);
```

For more information about naming and lookups, see "Accessing the Naming Context" on page 239.

3 **Define the `ejb-ref` elements in the `application-client.xml` file and the corresponding `sun-application-client.xml` file.**

For more information on the `sun-application-client.xml` file, see "The sun-application-client.xml file" on page 297. For a general explanation of how to map JNDI names using reference elements, see "Mapping References" on page 243.

4 **Deploy the application client and EJB component together in an application.**

For more information on deployment, see "Tools for Deployment" on page 93. To get the client JAR file, use the `--retrieve` option.

To retrieve the stubs and ties whether or not you requested their generation during deployment, use the `asadmin get-client-stubs` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

5 **Ensure that the client JAR file includes the following files:**

- a Java class to access the bean
- `application-client.xml` - J2EE 1.4 application client deployment descriptor.
- `sun-application-client.xml` - Application Server specific client deployment descriptor. For information on the `sun-application-client.xml` file, see "The sun-application-client.xml file" on page 297.
- The `MANIFEST.MF` file. This file contains the main class, which states the complete package prefix and class name of the Java client.

You can package the application client using the `package-appclient` script. This is optional. See "Packaging an Application Client Using the ACC" on page 193.

6 **Copy the following JAR files to the client machine and include them in the classpath on the client side:**

- `appserv-rt.jar` - available at *install-dir*`/lib`
- `j2ee.jar` - available at *install-dir*`/lib`
- The client JAR file

7 **To access EJB components that are residing in a remote system, make the following changes to the `sun-acc.xml` file:**

- Define the "target-server" on page 401 element's `address` attribute to reference the remote server machine.
- Define the "target-server" on page 401 element's `port` attribute to reference the ORB port on the remote server.

This information can be obtained from the `domain.xml` file on the remote system. For more information on `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

For more information about the `sun-acc.xml` file, see "The sun-acc.xml File" on page 298.

8 **Run the application client. See "Running an Application Client Using the ACC" on page 193.**

▼ To access a JMS resource from an application client

1 **Create a JMS client.**

For detailed instructions on developing a JMS client, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JMS.html#wp84181`.

2 **Next, configure a JMS resource on the Application Server.**

For information on configuring JMS resources, see "Creating JMS Resources: Destinations and Connection Factories" on page 249.

3 **Define the `resource-ref` elements in the `application-client.xml` file and the corresponding `sun-application-client.xml` file.**

For more information on the `sun-application-client.xml` file, see "The sun-application-client.xml file" on page 297. For a general explanation of how to map JNDI names using reference elements, see "Mapping References" on page 243.

4 **Ensure that the client JAR file includes the following files:**

- A Java class to access the resource.
- `application-client.xml` - J2EE 1.4 application client deployment descriptor.
- `sun-application-client.xml` - Application Server specific client deployment descriptor. For information on the `sun-application-client.xml` file, see "The sun-application-client.xml file" on page 297.
- The `MANIFEST.MF` file. This file contains the main class, which states the complete package prefix and class name of the Java client.

You can package the application client using the `package-appclient` script. This is optional. See "Packaging an Application Client Using the ACC" on page 193.

5 **Copy the following JAR files to the client machine and include them in the classpath on the client side:**

- `appserv-rt.jar` - available at *install-dir*`/lib`
- `j2ee.jar` - available at *install-dir*`/lib`
- `imqjmsra.jar` - available at *install-dir*`/lib/install/aplications/jmsra`
- The client JAR file

6 **Run the application client.**

See "Running an Application Client Using the ACC" on page 193.

Running an Application Client Using the ACC

To run an application client, launch the ACC using the `appclient` script. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Packaging an Application Client Using the ACC

The `package-appclient` script, located in the *install-dir*`/bin` directory, is used to package a client application into a single `appclient.jar` file. Packaging an application client involves the following main steps:

Editing the Configuration File

Modify the environment variables in `asenv.conf` file located in the *install-dir*`/config` directory as shown below:

- `$AS_INSTALL` to reference the location where the package was un-jared plus `/appclient`. For example: `$AS_INSTALL=/`*install-dir*`/appclient`.
- `$AS_NSS` to reference the location of the NSS libraries. For example:

  UNIX:

  `$AS_NSS=/`*install-dir*`/appclient/lib`

  WINDOWS:

  `%AS_NSS%=\`*install-dir*`\appclient\bin`
- `$AS_JAVA` to reference the location where the JDK is installed.
- `$AS_ACC_CONFIG` to reference the configuration XML file (`sun-acc.xml`). The `sun-acc.xml` is located at *install-dir*`/config`.
- `$AS_IMQ_LIB` to reference the imq home. Use *domain-dir*`/imq/lib`.

Editing the appclient Script

Modify the `appclient` script file as follows:

UNIX:

Change `$CONFIG_HOME/asenv.conf` to *your-ACC-dir*`/config/asenv.conf`.

Windows:

Change `%CONFIG_HOME%\config\asenv.bat` to *your-ACC-dir*`\config\asenv.bat`

Editing the sun-acc.xml File

Modify `sun-acc.xml` file to set the following attributes:

- Ensure that the `DOCTYPE` references *install-dir*`/lib/dtds` to *your-ACC-dir*`/lib/dtds`.
- Ensure that the `<target-server>` `address` attribute references the remote server machine.
- Ensure that the `<target-server>` `port` attribute references the ORB port on the remote server.
- To log the messages in a file, specify a file name for the `log-service` element's `file` attribute. You can also set the log level. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE client-container SYSTEM
"file:install-dir/lib/dtds/sun-application-client-container_1_0.dtd">
<client-container>
    <target-server name="qasol-e1" address="qasol-e1" port="3700">
    <log-service level="WARNING"/>
</client-container>
```

For more information on the `sun-acc.xml` file, see "The sun-acc.xml File" on page 298.

Setting Security Options

You can run the application client using SSL with certificate authentication. To set the security options, modify the `sun-acc.xml` file as shown in the code illustration below. For more information on the `sun-acc.xml` file, see "The sun-acc.xml File" on page 298.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE client-container SYSTEM
"file:install-dir/lib/dtds/sun-application-client-container_1_0.dtd">
<client-container>
  <target-server name="qasol-e1" address="qasol-e1" port="3700">
    <security>
      <ssl cert-nickname="cts"
        ssl2-enabled="false"
```

```
        ssl2-ciphers="-rc4,-rc4export,-rc2,-rc2export,-des,-desede3"
        ssl3-enabled="true"
        ssl3-tls-ciphers="+rsa_rc4_128_md5,-rsa_rc4_40_md5,+rsa3_des_sha,
            +rsa_des_sha,-rsa_rc2_40_md5,-rsa_null_md5,-rsa_des_56_sha,
            -rsa_rc4_56_sha"
        tls-enabled="true"
        tls-rollback-enabled="true"/>
      <cert-db path="ignored" password="ignored"/> <!-- not used -->
    </security>
  </target-server>
  <client-credential user-name="j2ee" password="j2ee"/>
  <log-service level="WARNING"/>
</client-container>
```

▼ To use the `package-appclient` script bundled with the Application Server

1 **Under *install-dir* `/bin` directory, run the `package-appclient` script.**

For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

This creates an `appclient.jar` file and stores it under *install-dir*`/lib/appclient/` directory.

Note – The `appclient.jar` file provides an application client container package targeted at remote hosts and does not contain a server installation. You can run this file from a remote machine with the same operating system as where it is created. That is, `appclient.jar` created on a Solaris platform does not function on Windows.

---

2 **Copy the *install-dir* `/lib/appclient/appclient.jar` file to the desired location.**

The `appclient.jar` file contains the following files:

- `appclient/bin` - contains the `appclient` script used to launch the ACC.
- `appclient/lib` - contains the JAR and runtime shared library files.
- `appclient/lib/appclient` - contains the following files:
  - `sun-acc.xml` - the ACC configuration file.
  - "client.policy" on page 196 file- the security manager policy file for the ACC.
  - `appclientlogin.conf` file - the login configuration file.
  - `client.jar` file - created during the deployment of the client application.
- `appclient/lib/dtds` - contains `sun-application_client-container_1_0.dtd`, which is the DTD corresponding to `sun-acc.xml`.

client.policy

The `client.policy` file is the J2SE policy file used by the application client. Each application client has a `client.policy` file. The default policy file limits the permissions of J2EE deployed application clients to the minimal set of permissions required for these applications to operate correctly. If an application client requires more than this default set of permissions, edit the `client.policy` file to add the custom permissions that your application client needs. Use the J2SE standard policy tool or any text editor to edit this file.

For more information on using the J2SE policy tool, see `http://java.sun.com/docs/books/tutorial/security1.2/tour2/index.html`.

For more information about the permissions you can set in the `client.policy` file, see `http://java.sun.com/j2se/1.4/docs/guide/security/permissions.html`.

Developing Clients Without the ACC

This section describes the procedure to create, assemble, and deploy a Java-based client that is not packaged using the Application Client Container (ACC). This section describes the following topics:

- "To access an EJB component from a stand-alone client" on page 196
- "To access an EJB component from a server-side module" on page 197
- "To access a JMS resource from a stand-alone client" on page 198

For information about using the ACC, see "Developing Clients Using the ACC" on page 190.

▼ To access an EJB component from a stand-alone client

1 In your client code, instantiate the `InitialContext`:

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

It is not necessary to explicitly instantiate a naming context that points to the CosNaming service.

2 In the client code, look up the home object by specifying the JNDI name of the home object.

For example:

```
Object ref = ctx.lookup("jndi-name");
BeanAHome = (BeanAHome)PortableRemoteObject.narrow(ref,BeanAHome.class);
```

For more information about naming and lookups, see "Accessing the Naming Context" on page 239.

3. **Deploy the EJB component to be accessed.**

   For more information on deployment, see "Tools for Deployment" on page 93.

4. **Copy the following JAR files to the client machine and include them in the classpath on the client side:**

   - `appserv-rt.jar` - available at *install-dir*`/lib`
   - `j2ee.jar` - available at *install-dir*`/lib`

5. **To access EJB components that are residing in a remote system, set the values for the Java Virtual Machine startup options:**

   ```
   jvmarg value = "-Dorg.omg.CORBA.ORBInitialHost=${ORBhost}"
   jvmarg value = "-Dorg.omg.CORBA.ORBInitialPort=${ORBport}"
   ```

   Here *ORBhost* is the Application Server hostname and *ORBport* is the ORB port number (default is `3700`).

   This information can be obtained from the `domain.xml` file on the remote system. For more information on `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

6. **Run the stand-alone client.**

   As long as the client environment is set appropriately and the JVM is compatible, you merely need to run the `main` class.

▼ To access an EJB component from a server-side module

A server-side module can be a servlet, another EJB component, or another type of module.

1. **In your module code, instantiate the `InitialContext`:**

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

   It is not necessary to explicitly instantiate a naming context that points to the CosNaming service.

2. **In the module code, look up the home object by specifying the JNDI name of the home object. For example:**

   ```
   Object ref = ctx.lookup("jndi-name");
   BeanAHome = (BeanAHome)PortableRemoteObject.narrow(ref,BeanAHome.class);
   ```

   For more information about naming and lookups, see "Accessing the Naming Context" on page 239.

3 **Deploy the EJB component to be accessed.**

For more information on deployment, see "Tools for Deployment" on page 93.

4 **To access EJB components that are residing in a remote system, set the values for the Java Virtual Machine startup options:**

`jvmarg value = "-Dorg.omg.CORBA.ORBInitialHost=${`*ORBhost*`}"jvmarg value = "-Dorg.omg.CORBA.ORBInitialPort=${`*ORBport*`}"`

Here *ORBhost* is the Application Server hostname and *ORBport* is the ORB port number (default is `3700`).

This information can be obtained from the `domain.xml` file on the remote system. For more information on `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

5 **Deploy the module.**

For more information on deployment, see "Tools for Deployment" on page 93.

▼ To access a JMS resource from a stand-alone client

1 **Create a JMS client.**

For detailed instructions on developing a JMS client, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JMS.html#wp84181`.

2 **Next, configure a JMS resource on the Application Server.**

For information on configuring JMS resources, see "Creating JMS Resources: Destinations and Connection Factories" on page 249.

3 **Copy the following JAR files to the client machine and include them in the classpath on the client side:**

- `appserv-rt.jar` - available at *install-dir*`/lib`
- `j2ee.jar` - available at *install-dir*`/lib`
- `imqjmsra.jar` - available at *install-dir*`/lib/install/aplications/jmsra`

4 **Set the values for the Java Virtual Machine startup options:**

`jvmarg value = "-Dorg.omg.CORBA.ORBInitialHost=${`*ORBhost*`}"`
`jvmarg value = "-Dorg.omg.CORBA.ORBInitialPort=${`*ORBport*`}"`

Here *ORBhost* is the Application Server hostname and *ORBport* is the ORB port number (default is `3700`).

This information can be obtained from the `domain.xml` file. For more information on `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

5 Run the stand-alone client.

As long as the client environment is set appropriately and the JVM is compatible, you merely need to run the `main` class.

◆ ◆ ◆ CHAPTER 9

Developing Connectors

This chapter describes Sun Java System Application Server support for the J2EE Connector 1.5 architecture.

The J2EE Connector architecture provides a Java solution to the problem of connectivity between multiple application servers and existing enterprise information systems (EISs). By using the J2EE Connector architecture, EIS vendors no longer need to customize their product for each application server. Application server vendors who conform to the J2EE Connector architecture do not need to write custom code to add connectivity to a new EIS.

This chapter uses the terms *connector* and *resource adapter* interchangeably. Both terms refer to a resource adapter module that is developed in conformance with the J2EE Connector Specification 1.5.

For more information about connectors, see the J2EE Connector architecture home page, at `http://java.sun.com/j2ee/connector/`.

For connector examples, see http://developers.sun.com/prodtech/appserver/reference/techart/as8_connectors.

This chapter includes the following topics:

Connector 1.5 Support in the Application Server

The Application Server supports the development and deployment of resource adapters that are compatible with Connector 1.5 specification (and, for backward compatibility, the Connector 1.0 specification).

The Connector 1.0 specification defines the outbound connectivity system contracts between the resource adapter and the Application Server. The Connector 1.5 specification introduces major additions in defining system level contracts between the Application Server and the resource adapter with respect to the following:

- Inbound connectivity from an EIS - The Connector 1.5 defines the transaction and message inflow system contracts for achieving inbound connectivity from an EIS. The message inflow contract also serves as a standard message provider pluggability contract, thereby allowing various providers of messaging systems to seamlessly plug in their products with any application server that supports the message inflow contract.
- Resource adapter life cycle management and thread management - These features are available through the lifecycle and work management contracts.

Connector Architecture for JMS and JDBC

In the Administration Console, connector, JMS, and JDBC resources are handled differently, but they use the same underlying Connector architecture. In the Application Server, all communication to an EIS, whether to a message provider or an RDBMS, happens through the Connector architecture. To provide JMS infrastructure to clients, the Application Server uses the Sun Java System Message Queue software. To provide JDBC infrastructure to clients, the Application Server uses its own JDBC system resource adapters. The application server automatically makes these system resource adapters available to any client that requires them.

For more information about JMS in the Application Server, see Chapter 14, “Using the Java Message Service.” For more information about JDBC in the Application Server, see Chapter 11, “Using the JDBC API for Database Access.”

Connector Configuration

The Application Server does not need to use `sun-ra.xml`, which previous Application Server versions used, to store server-specific deployment information inside a Resource Adapter Archive (RAR) file. (However, the `sun-ra.xml` file is still supported for backward compatibility.) Instead, the information is stored in the server configuration. As a result, you can create multiple connector connection pools for a connection definition in a functional resource adapter instance, and you can create multiple user-accessible connector resources (that is, registering a resource with a JNDI name) for a connector connection pool. In addition, dynamic changes can be made to connector connection pools and the connector resource properties without restarting the Application Server.

Deploying and Configuring a Stand-Alone Connector Module

You can deploy a stand-alone connector module using the Administration Console or the `asadmin` command. For information about using the Administration Console, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*. For information about using the `asadmin` command, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Deploying a stand-alone connector module allows multiple deployed J2EE applications to share the connector module. A resource adapter configuration is automatically created for the connector module.

▼ To deploy and configure a stand-alone connector module

1. **Deploy the connector module in one of the following ways.**
   - In the Administration Console, open the Applications component and select Connector Modules. When you deploy the connector module, a resource adapter configuration is automatically created for the connector module.
   - Use the `asadmin deploy` or `asadmin deploydir` command. To override the default configuration properties of a resource adapter, if necessary, use the `asadmin create-resource-adapter-config` command.
2. **Configure connector connection pools for the deployed connector module in one of the following ways:**
   - In the Administration Console, open the Resources component, select Connectors, and select Connector Connection Pools.
   - Use the `asadmin create-connector-connection-pool` command.
3. **Configure connector resources for the connector connection pools in one of the following ways.**
   - In the Administration Console, open the Resources component, select Connectors, and select Connector Resources.
   - Use the `asadmin create-connector-resource` command.

   This associates a connector resource with a JNDI name.
4. **Create an administered object for an inbound resource adapter, if necessary, in one of the following ways:**
   - In the Administration Console, open the Resources component, select Connectors, and select Admin Object Resources.

- Use the `asadmin create-admin-object` command.

Redeploying a Stand-Alone Connector Module

Redeployment of a connector module maintains all connector connection pools, connector resources, and administered objects defined for the previously deployed connector module. You need not reconfigure any of these resources.

However, you should redeploy any dependent modules. A dependent module uses or refers to a connector resource of the redeployed connector module. Redeployment of a connector module results in the shared class loader reloading the new classes. Other modules that refer to the old resource adapter classes must be redeployed to gain access to the new classes. For more information about classloaders, see "Classloaders" on page 78.

During connector module redeployment, the server log provides a warning indicating that all dependent applications should be redeployed. Client applications or application components using the connector module's resources may throw class cast exceptions if dependent applications are not redeployed after connector module redeployment.

To disable automatic redeployment, set the `--force` option to `false`. In this case, if the connector module has already been deployed, the Application Server provides an error message.

Deploying and Configuring an Embedded Resource Adapter

A connector module can be deployed as a J2EE component in a J2EE application. Such connectors are only visible to components residing in the same J2EE application. Simply deploy this J2EE application as you would any other J2EE application.

You can create new connector connection pools and connector resources for a connector module embedded within a J2EE application by prefixing the connector name with *app-name*#. For example, if an application `appX.ear` has `jdbcra.rar` embedded within it, the connector connection pools and connector resources refer to the connector module as `appX#jdbcra`.

However, an embedded connector module cannot be undeployed using the name *app-name*#*connector-name*. To undeploy the connector module, you must undeploy the application in which it is embedded.

The association between the physical JNDI name for the connector module in the Application Server and the logical JNDI name used in the application component is specified in the Application Server specific XML descriptor `sun-ejb-jar.xml`. You can either hand code this association or use the deploytool to make this association. (For more information about using the deploytool, see "deploytool" on page 43.)

Advanced Connector Configuration Options

You can use these advanced connector configuration options:

Thread Pools

Connectors can submit work instances to the Application Server for execution. By default, the Application Server services work requests for all connectors from its default thread pool. However, you can associate a specific user-created thread pool to service work requests from a connector. A thread pool can service work requests from multiple resource adapters. To create a thread pool:

- In the Administration Console, select Thread Pools under the relevant configuration. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-threadpool` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

To associate a connector with a thread pool:

- In the Administration Console, open the Applications component and select Connector Modules. Deploy the module, or select the previously deployed module. Specify the name of the thread pool in the Thread Pool ID field. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `--threadpoolid` option of the `asadmin create-resource-adapter-config` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

If you create a resource adapter configuration for a connector module that is already deployed, the connector module deployment is restarted with the new configuration properties.

Security Maps

Create a security map for a connector connection pool to map an application principal or a user group to a back end EIS principal. The security map is usually used in situations where one or more EIS back end principals are used to execute operations (on the EIS) initiated by various principals or user groups in the application.

To create or update security maps for a connector connection pool:

- In the Administration Console, open the Resources component, select Connectors, select Connector Connection Pools, and select the Security Maps tab. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-connector-security-map` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual.*

If a security map already exists for a connector connection pool, the new security map is appended to the previous one. The connector security map configuration supports the use of the wildcard asterisk (*) to indicate all users or all user groups.

When an application principal initiates a request to an EIS, the Application Server first checks for an exact match to a mapped back end EIS principal using the security map defined for the connector connection pool. If there is no exact match, the Application Server uses the wild card character specification, if any, to determined the mapped back end EIS principal.

Overriding Configuration Properties

You can override the properties specified in the `ra.xml` file of a resource adapter. Use the `asadmin create-resource-adapter-config` command to create a configuration for a resource adapter. Use this command's `--property` option to specify a name-value pair for a resource adapter property.

You can use the `asadmin create-resource-adapter-config` command either before or after resource adapter deployment. If it is executed after deploying the resource adapter, the existing resource adapter is restarted with the new properties. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Testing a Connection Pool

After configuring a connector connection pool, use the `asadmin ping-connection-pool` command to test the health of the underlying connections. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Handling Invalid Connections

If a resource adapter generates a `ConnectionErrorOccured` event, the Application Server considers the connection invalid and removes the connection from the connection pool. Typically, a resource adapter generates a `ConnectionErrorOccured` event when it finds a `ManagedConnection` object unusable. Reasons can be network failure with the EIS, EIS failure, fatal problems with resource adapter, and so on. If the `fail-all-connections` property in the connection pool configuration is set to `true`, all connections are destroyed and the pool is recreated.

You can set the `fail-all-connections` configuration property during creation of a connector connection pool. Or, you can use the `asadmin set` command to dynamically reconfigure a previously set property. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

The interface `ValidatingManagedConnectionFactory` exposes the method `getInvalidConnections` to allow retrieval of the invalid connections. The Application Server checks if the resource adapter implements this interface, and if it does, invalid connections are removed when the connection pool is resized.

Setting the Shutdown Timeout

According to the Connector 1.5 specification, while an application server shuts down, all resource adapters should be stopped. A resource adapter might hang during shutdown, since shutdown is typically a resource intensive operation. To avoid such a situation, you can set a timeout that aborts resource adapter shutdown if exceeded. The default timeout is 30 seconds per resource adapter module. To configure this timeout:

- In the Administration Console, select JMS/Connector Service under the relevant configuration. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the following command:

  ```
  asadmin set server-instance.connector-service.shutdown-timeout-in-seconds="num-secs"
  ```

  For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

The Application Server deactivates all message-driven bean deployments before stopping a resource adapter.

Using Last Agent Optimization of Transactions

Transactions that involve multiple resources or multiple participant processes are *distributed* or *global* transactions. A global transaction can involve one non-XA resource if last agent optimization is enabled. Otherwise, all resources must be XA. For more information about transactions in the Application Server, see Chapter 12, "Using the Transaction Service."

The Connector 1.5 specification requires that if a resource adapter supports `XATransaction`, the `ManagedConnection` created from that resource adapter must support both distributed and local transactions. Therefore, even if a resource adapter supports `XATransaction`, you can configure its connector connection pools as non-XA or without transaction support for better performance. A non-XA resource adapter becomes the last agent in the transactions in which it participates.

The value of the connection pool configuration property `transaction-support` defaults to the value of the `transaction-support` property in the `ra.xml` file. The connection pool configuration property can override the `ra.xml` file property if the transaction level in the connection pool configuration property is lower. If the value in the connection pool configuration property is higher, it is ignored.

Inbound Communication Support

The Connector 1.5 specification defines the transaction and message inflow system contracts for achieving inbound connectivity from an EIS. The message inflow contract also serves as a standard message provider pluggability contract, thereby allowing various message providers to seamlessly plug in their products with any application server that supports the message inflow contract. In the inbound communication model, the EIS initiates all communication to an application. An application can be composed of enterprise beans (session, entity, or message-driven beans), which reside in an EJB container.

Incoming messages are received through a message endpoint, which is a message-driven bean. This message-driven bean asynchronously consumes messages from a message provider. An application can also synchronously send and receive messages directly using messaging style APIs.

A resource adapter supporting inbound communication provides an instance of an `ActivationSpec` JavaBean class for each supported message listener type. Each class contains a set of configurable properties that specify endpoint activation configuration information during message-driven bean deployment. The `required-config-property` element in the `ra.xml` file provides a list of configuration property names required for each activation specification. An endpoint activation fails if the required property values are not specified. Values for the properties that are overridden in the message-driven bean's deployment descriptor are applied to the `ActivationSpec` JavaBean when the message-driven bean is deployed.

Administered objects can also be specified for a resource adapter, and these JavaBeans are specific to a messaging style or message provider. For example, some messaging styles may need applications to use special administered objects (such as `Queue` and `Topic` objects in JMS). Applications use these objects to send and synchronously receive messages using connection objects using messaging style APIs. For more information about administered objects, see Chapter 14, "Using the Java Message Service."

Configuring a Message Driven Bean to Use a Resource Adapter

The Connectors 1.5 specification's message inflow contract provides a generic mechanism to plug in a wide-range of message providers, including JMS, into a J2EE-compatible application server. Message providers use a resource adapter and dispatch messages to message endpoints, which are implemented as message-driven beans.

The message-driven bean developer provides activation configuration information in the message-driven bean's `ejb-jar.xml` file. Configuration information includes messaging-style-specific configuration details, and possibly message-provider-specific details as well. The message-driven bean deployer uses this configuration information to set up the activation specification JavaBean. The activation configuration properties specified in `ejb-jar.xml` override configuration properties in the activation specification definition in the `ra.xml` file.

According to the EJB specification, the messaging-style-specific descriptor elements contained within the activation configuration element are not specified because they are specific to a messaging provider. In the following sample message-driven bean `ejb-jar.xml`, a message-driven bean has the following activation configuration property names: `destinationType`, `SubscriptionDurability`, and `MessageSelector`.

```
<!--  A sample MDB that listens to a JMS Topic -->
<!-- message-driven bean deployment descriptor -->
...
 <activation-config>
   <activation-config-property>
     <activation-config-property-name>
       destinationType
     </activation-config-property-name>
     <activation-config-property-value>
       javax.jms.Topic
     </activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
     <activation-config-property-name>
       SubscriptionDurability
     </activation-config-property-name>
     <activation-config-property-value>
       Durable
     </activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
     <activation-config-property-name>
       MessageSelector
     </activation-config-property-name>
     <activation-config-property-value>
       JMSType = 'car' AND color = 'blue'
```

```
    </activation-config-property-value>
  </activation-config-property>
 ...
 </activation-config>
...
```

When the message-driven bean is deployed, the value for the `resource-adapter-mid` element in the `sun-ejb-jar.xml` file is set to the resource adapter module name that delivers messages to the message endpoint (to the message-driven bean). In the following example, the `jmsra` JMS resource adapter, which is the bundled resource adapter for the Sun Java System Message Queue message provider, is specified as the resource adapter module identifier for the `SampleMDB` bean.

```
<sun-ejb-jar>
<enterprise-beans>
    <unique-id>1</unique-id>
    <ejb>
       <ejb-name>SampleMDB</ejb-name>
       <jndi-name>SampleQueue</jndi-name>
    <!-- JNDI name of the destination from which messages would be
         delivered from MDB needs to listen to -->
    ...
    </ejb>
    <mdb-resource-adapter>
    <resource-adapter-mid>jmsra</resource-adapter-mid>
    <!-- Resource Adapter Module Id that would deliver messages to
         this message endpoint -->
    </mdb-resource-adapter>
...
</sun-ejb-jar>
```

When the message-driven bean is deployed, the Application Server uses the `resourceadapter-mid` setting to associate the resource adapter with a message endpoint through the message inflow contract. This message inflow contract with the application server gives the resource adapter a handle to the `MessageEndpointFactory` and the `ActivationSpec` JavaBean, and the adapter uses this handle to deliver messages to the message endpoint instances (which are created by the `MessageEndpointFactory`).

When a message-driven bean first created for use on the Application Server 7 is deployed, the Connector runtime transparently transforms the previous deployment style to the current connector-based deployment style. If the deployer specifies neither a `resource-adapter-mid` property nor the Message Queue resource adapter's activation configuration properties, the Connector runtime maps the message-driven bean to the `jmsra` system resource adapter and converts the JMS-specific configuration to the Message Queue resource adapter's activation configuration properties.

Example Resource Adapter for Inbound Communication

The inbound sample connector bundled with the Application Server is a good example of an application utilizing the inbound connectivity contract of the J2EE Connector Architecture 1.5 specification. This sample connector is available at *install-dir*`/samples/connectors/apps/mailconnector`.

This example connector shows how to create an inbound J2EE Connector Architecture 1.5-compliant resource adapter and deploy its components. It shows how these resource adapters interact with other application components. The inbound sample resource adapter allows message endpoints (that is, message-driven beans) to receive email messages delivered to a specific mailbox folder on a given mail server.

The application that is bundled along with this inbound sample connector provides a simple Remote Method Invocation (RMI) back end service that allows the user to monitor the mailbox folders specified by the message-driven beans. The sample application also contains a sample message-driven bean that illustrates how the activation configuration specification properties of the message-driven bean provide the configuration parameters that the back end and resource adapter require to monitor a specific mailbox folder.

The `onMessage` method of the message-driven bean uses the JavaMail API to send a reply acknowledging the receipt of the message. This reply is sufficient to verify that the full process is working.

◆ ◆ ◆ CHAPTER 10

Developing Lifecycle Listeners

Lifecycle listener modules provide a means of running short or long duration Java-based tasks within the application server environment, such as instantiation of singletons or RMI servers. These modules are automatically initiated at server startup and are notified at various phases of the server life cycle.

All lifecycle module classes and interfaces are in the *install-dir*`/lib/appserv-rt.jar` file.

The following sections describe how to create and use a lifecycle listener module:

Server Life Cycle Events

A lifecycle module listens for and performs its tasks in response to the following events in the server life cycle:

- After the `INIT_EVENT`, the server reads the configuration, initializes built-in subsystems (such as security and logging services), and creates the containers.
- After the `STARTUP_EVENT`, the server loads and initializes deployed applications.
- After the `READY_EVENT`, the server is ready to service requests.
- After the `SHUTDOWN_EVENT`, the server destroys loaded applications and stops.
- After the `TERMINATION_EVENT`, the server closes the containers, the built-in subsystems, and the server runtime environment.

These events are defined in the `LifecycleEvent` class.

The lifecycle modules that listen for these events implement the `LifecycleListener` interface.

The LifecycleListener Interface

To create a lifecycle module is to configure a customized class that implements the `com.sun.appserv.server.LifecycleListener` interface. You can create and simultaneously execute multiple lifecycle modules.

The `LifecycleListener` interface defines this method:

```
public void handleEvent(com.sun.appserv.server.LifecycleEvent event)
throws ServerLifecycleException
```

This method responds to a lifecycle event and throws a `com.sun.appserv.server.ServerLifecycleException` if an error occurs.

A sample implementation of the `LifecycleListener` interface is the `LifecycleListenerImpl.java` file, which you can use for testing lifecycle events.

The LifecycleEvent Class

The `com.sun.appserv.server.LifecycleEvent` class defines a server life cycle event. The following methods are associated with the event:

- `public java.lang.Object getData()`

  This method returns the data associated with the event.

- `public int getEventType()`

  This method returns the type of the last event, which is `INIT_EVENT`, `STARTUP_EVENT`, `READY_EVENT`, `SHUTDOWN_EVENT`, or `TERMINATION_EVENT`.

- `public com.sun.appserv.server.LifecycleEventContext getLifecycleEventContext()`

  This method returns the lifecycle event context, described next.

A `LifecycleEvent` instance is passed to the `LifecycleListener.handleEvent` method.

The Server Lifecycle Event Context

The `com.sun.appserv.server.LifecycleEventContext` interface exposes runtime information about the server. The lifecycle event context is created when the `LifecycleEvent` class is instantiated at server initialization. The `LifecycleEventContext` interface defines these methods:

- `public java.lang.String[] getCmdLineArgs()`

  This method returns the server startup command-line arguments.
- `public java.lang.String getInstallRoot()`

  This method returns the server installation root directory.
- `public java.lang.String getInstanceName()`

  This method returns the server instance name.
- `public javax.naming.InitialContext getInitialContext()`

  This method returns the initial JNDI naming context. The naming environment for lifecycle modules is installed after the `STARTUP_EVENT`. A lifecycle module can look up any resource by its `jndi-name` attribute after the `READY_EVENT`.

If a lifecycle module needs to look up resources, it can do so after the `READY_EVENT`. It can use the `getInitialContext()` method to get the initial context to which all the resources are bound.

Deploying a Lifecycle Module

You can deploy a lifecycle module using the following tools:

- In the Administration Console, open the Applications component and go to the Lifecycle Modules page. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin` `create-lifecycle-module` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

You do not need to specify a classpath for the lifecycle module if you place it in the *domain-dir*`/lib` or *domain-dir*`/lib/classes` directory.

After you deploy a lifecycle module, you must restart the server to activate it. The server instantiates it and registers it as a lifecycle event listener at server initialization.

Note – If the `is-failure-fatal` setting is set to `true` (the default is `false`), lifecycle module failure prevents server initialization or startup, but not shutdown or termination.

Considerations for Lifecycle Modules

The resources allocated at initialization or startup should be freed at shutdown or termination. The lifecycle module classes are called synchronously from the main server thread, therefore it is important to ensure that these classes don't block the server. Lifecycle modules can create threads if appropriate, but these threads must be stopped in the shutdown and termination phases.

The LifeCycleModule Classloader is the parent class loader for lifecycle modules. Each lifecycle module's `classpath` in `domain.xml` is used to construct its class loader. All the support classes needed by a lifecycle module must be available to the LifeCycleModule Classloader or its parent, the Connector Classloader.

You must ensure that the `server.policy` file is appropriately set up, or a lifecycle module trying to perform a `System.exec()` might cause a security access violation. For details, see "The server.policy File" on page 54.

The configured properties for a lifecycle module are passed as properties after the `INIT_EVENT`. The JNDI naming context is not available before the `STARTUP_EVENT`. If a lifecycle module requires the naming context, it can get this after the `STARTUP_EVENT`, `READY_EVENT`, or `SHUTDOWN_EVENT`.

PART III

Using Services and APIs

◆ ◆ ◆ CHAPTER 11

Using the JDBC API for Database Access

This chapter describes how to use the Java™ Database Connectivity (JDBC™) API for database access with the Sun Java System Application Server. This chapter also provides high level JDBC implementation instructions for servlets and EJB™ components using the Application Server. The Application Server supports the JDBC 3.0 API, which encompasses the JDBC 2.0 Optional Package API.

The JDBC specifications are available at
`http://java.sun.com/products/jdbc/download.html`.

A useful JDBC tutorial is located at
`http://java.sun.com/docs/books/tutorial/jdbc/index.html`.

For explanations of two-tier and three-tier database access models, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Note – The Application Server does not support connection pooling or transactions for an application's database access if it does not use standard J2EE™ `DataSource` objects.

This chapter discusses the following topics:

General Steps for Creating a JDBC Resource

To prepare a JDBC resource for use in J2EE applications deployed to the Application Server, perform the following tasks:

- "Integrating the JDBC Driver" on page 220
- "Creating a Connection Pool" on page 220
- "Testing a Connection Pool" on page 221
- "Creating a JDBC Resource" on page 221

For information about how to configure some specific JDBC drivers, see the "Configurations for Specific JDBC Drivers" on page 224.

Integrating the JDBC Driver

To use JDBC features, you must choose a JDBC driver to work with the Application Server, then you must set up the driver. This section covers these topics:

- "Supported Database Drivers" on page 220
- "Making the JDBC Driver JAR Files Accessible" on page 220

Supported Database Drivers

Supported JDBC drivers are those that have been fully tested by Sun. For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

Note – Because the drivers and databases supported by the Application Server are constantly being updated, and because database vendors continue to upgrade their products, always check with Sun technical support for the latest database support information.

Making the JDBC Driver JAR Files Accessible

To integrate the JDBC driver into a Application Server domain, copy the JAR files into the *domain-dir*`/lib/ext` directory, then restart the server. This makes classes accessible to any application or module across the domain. For more information about Application Server classloaders, see "Classloaders" on page 78.

Creating a Connection Pool

When you create a connection pool that uses JDBC technology (a *JDBC connection pool*) in the Application Server, you can define many of the characteristics of your database connections.

You can create a JDBC connection pool in one of these ways:

- In the Administration Console, open the Resources component, open the JDBC component, and select Connection Pools. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-jdbc-connection-pool` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Testing a Connection Pool

You can test a JDBC connection pool for usability in one of these ways:

- In the Administration Console, open the Resources component, open the JDBC component, select Connection Pools, and select the connection pool you want to test. Then select the Ping button in the top right corner of the page. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin ping-connection-pool` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*

Both these commands fail and display an error message unless they successfully connect to the connection pool.

Creating a JDBC Resource

A JDBC resource, also called a data source, lets you make connections to a database using `getConnection()`. Create a JDBC resource in one of these ways:

- In the Administration Console, open the Resources component, open the JDBC component, and select JDBC Resources. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-jdbc-resource` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Creating Applications That Use the JDBC API

An application that uses the JDBC API is an application that looks up and connects to one or more databases. This section covers these topics:

Sharing Connections

When multiple connections acquired by an application use the same JDBC resource, the connection pool provides connection sharing within the same transaction scope. For example, suppose Bean A starts a transaction and obtains a connection, then calls a method in Bean B. If Bean B acquires a connection to the same JDBC resource with the same sign-on information, and if Bean A completes the transaction, the connection can be shared.

Connections obtained through a resource are shared only if the resource reference declared by the J2EE component allows it to be shareable. This is specified in a component's deployment descriptor by setting the `res-sharing-scope` element to `Shareable` for the particular resource reference. To turn off connection sharing, set `res-sharing-scope` to `Unshareable`.

For general information about connections and JDBC URLs, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Obtaining a Physical Connection from a Wrapped Connection

The `DataSource` implementation in the Application Server provides a `getConnection` method that retrieves the JDBC driver's `SQLConnection` from the Application Server's `Connection` wrapper. The method signature is as follows:

```
public java.sql.Connection getConnection(java.sql.Connection con)
throws java.sql.SQLException
```

For example:

```
InitialContext ctx = new InitialContext();
com.sun.appserv.DataSource ds = (com.sun.appserv.DataSource)
   ctx.lookup("jdbc/MyBase");
Connection con = ds.getConnection();
Connection drivercon = ds.getConnection(con);
// Do db operations.
con.close();
```

Using Non-Transactional Connections

The `DataSource` implementation in the Application Server provides a `getNonTxConnection` method, which retrieves a JDBC connection that is not in the scope of any transaction. There are two variants, as follows:

```
public java.sql.Connection getNonTxConnection() throws java.sql.SQLException
```

```
public java.sql.Connection getNonTxConnection(String user, String password)
throws java.sql.SQLException
```

Another way to get a non-transactional connection is to create a resource with the JNDI name ending in `__nontx`. This forces all connections looked up using this resource to be non transactional.

Typically, a connection is enlisted in the context of the transaction in which a `getConnection` call is invoked. However, a non-transactional connection is not enlisted in a transaction context even if a transaction is in progress.

The main advantage of using non-transactional connections is that the overhead incurred in enlisting and delisting connections in transaction contexts is avoided. However, use such connections carefully. For example, if a non-transactional connection is used to query the database while a transaction is in progress that modifies the database, the query retrieves the unmodified data in the database. This is because the in-progress transaction hasn't committed. For another example, if a non-transactional connection modifies the database and a transaction that is running simultaneously rolls back, the changes made by the non-transactional connection are not rolled back.

Here is a typical use case for a non-transactional connection: a component that is updating a database in a transaction context spanning over several iterations of a loop can refresh cached data by using a non-transactional connection to read data before the transaction commits.

Using JDBC Transaction Isolation Levels

For general information about transactions, see Chapter 12, "Using the Transaction Service"and the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*. For information about last agent optimization, which can improve performance, see "Transaction Scope" on page 236.

Not all database vendors support all transaction isolation levels available in the JDBC API. The Application Server permits specifying any isolation level your database supports. The following table defines transaction isolation levels.

TABLE 11–1 Transaction Isolation Levels

Transaction Isolation Level	Description
`TRANSACTION_READ_UNCOMMITTED`	Dirty reads, non-repeatable reads and phantom reads can occur.
`TRANSACTION_READ_COMMITTED`	Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
`TRANSACTION_REPEATABLE_READ`	Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
`TRANSACTION_SERIALIZABLE`	Dirty reads, non-repeatable reads and phantom reads are prevented.

Note that you cannot call `setTransactionIsolation()` during a transaction.

You can set the default transaction isolation level for a JDBC connection pool. For details, see "Creating a Connection Pool" on page 220.

To verify that a level is supported by your database management system, test your database programmatically using the `supportsTransactionIsolationLevel()` method in `java.sql.DatabaseMetaData`, as shown in the following example:

```
java.sql.DatabaseMetaData db;
if (db.supportsTransactionIsolationLevel(TRANSACTION_SERIALIZABLE)
   { Connection.setTransactionIsolation(TRANSACTION_SERIALIZABLE); }
```

For more information about these isolation levels and what they mean, see the JDBC 3.0 API specification.

Note – Applications that change the isolation level on a pooled connection programmatically risk polluting the pool, which can lead to errors.

Configurations for Specific JDBC Drivers

Application Server 8.2 is designed to support connectivity to any database management system with a corresponding JDBC driver. The following JDBC driver and database combinations are supported. These combinations have been tested with Application Server 8.2 and are found to be J2EE compatible. They are also supported for CMP.

- "Derby Type 4 Driver" on page 225
- "Sun Java System JDBC Driver for DB2 Databases" on page 226
- "Sun Java System JDBC Driver for Oracle 8i, 9i, and 10g Databases" on page 226
- "Sun Java System JDBC Driver for Microsoft SQL Server Databases" on page 227
- "Sun Java System JDBC Driver for Sybase Databases" on page 227
- "IBM DB2 8.1 Type 2 Driver" on page 228
- "JConnect Type 4 Driver for Sybase ASE 12.5 Databases" on page 228
- "MM MySQL Type 4 Driver (Non-XA)" on page 229

For an up to date list of currently supported JDBC drivers, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*.

Other JDBC drivers can be used with Application Server 8.2, but J2EE compliance tests have not been completed with these drivers. Although Sun offers no product support for these drivers, Sun offers limited support of the use of these drivers with Application Server 8.2.

- "MM MySQL Type 4 Driver (XA Only)" on page 229
- "Inet Oraxo JDBC Driver for Oracle 8i, 9i, and 10g Databases" on page 230
- "Inet Merlia JDBC Driver for Microsoft SQL Server Databases" on page 231
- "Inet Sybelux JDBC Driver for Sybase Databases" on page 231
- "Oracle Thin Type 4 Driver for Oracle 8i, 9i, and 10g Databases" on page 232

- "OCI Oracle Type 2 Driver for Oracle 8i, 9i, and 10g Databases" on page 233
- "IBM Informix Type 4 Driver" on page 234

For details about how to integrate a JDBC driver and how to use the Administration Console or the command line interface to implement the configuration, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Note – An Oracle database user running the `capture-schema` command needs ANALYZE ANY TABLE privileges if that user does not own the schema. These privileges are granted to the user by the database administrator. For information about `capture-schema`, see "Using the capture-schema Utility" on page 176.

Derby Type 4 Driver

The Derby JDBC driver is included with the Application Server by default, except for the Solaris bundled installation, which does not include Derby. Therefore, unless you have the Solaris bundled installation, you do not need to integrate this JDBC driver with the Application Server.

The JAR file for the Derby driver is `derbyclient.jar`.

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Derby`
- **DataSource Classname:** Specify one of the following:

  ```
  org.apache.derby.jdbc.ClientDataSource
  org.apache.derby.jdbc.ClientXADataSource
  ```

- **Properties:**
  - **`user`** - Specify the database user.

    This is only necessary if Derby is configured to use authentication. Derby does *not* use authentication by default. When the user is provided, it is the name of the schema where the tables reside.
  - **`password`** - Specify the database password.

    This is only necessary if Derby is configured to use authentication.
  - **`databaseName`** - Specify the name of the database.
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server if it is different from the default.

- **URL:** `jdbc:derby://`*serverName*`:`*portNumber*`/`*databaseName*`;create=true`

  Include the `;create=true` part only if you want the database to be created if it does not exist.

Sun Java System JDBC Driver for DB2 Databases

The JAR files for this driver are `smbase.jar`, `smdb2.jar`, and `smutil.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `DB2`
- **DataSource Classname:** `com.sun.sql.jdbcx.db2.DB2DataSource`
- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`databaseName`** - Set as appropriate.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
- **URL:** `jdbc:sun:db2://`*serverName*`:`*portNumber*`;databaseName=`*databaseName*

Sun Java System JDBC Driver for Oracle 8i, 9i, and 10g Databases

The JAR files for this driver are `smbase.jar`, `smoracle.jar`, and `smutil.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Oracle`
- **DataSource Classname:** `com.sun.sql.jdbcx.oracle.OracleDataSource`
- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`SID`** - Set as appropriate.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
- **URL:** `jdbc:sun:oracle://`*serverName*`[:`*portNumber*`][;SID=`*databaseName*`]`

Sun Java System JDBC Driver for Microsoft SQL Server Databases

The JAR files for this driver are `smbase.jar`, `smsqlserver.jar`, and `smutil.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `mssql`
- **DataSource Classname:** `com.sun.sql.jdbcx.sqlserver.SQLServerDataSource`
- **Properties:**
  - **`serverName`** - Specify the host name or IP address and the port of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`selectMethod`** - Set to `cursor`.
- **URL:** `jdbc:sun:sqlserver://`*serverName*`[:`*portNumber*`]`

Sun Java System JDBC Driver for Sybase Databases

The JAR files for this driver are `smbase.jar`, `smsybase.jar`, and `smutil.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Sybase`
- **DataSource Classname:** `com.sun.sql.jdbcx.sybase.SybaseDataSource`
- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`databaseName`** - Set as appropriate. This is optional.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
- **URL:** `jdbc:sun:sybase://`*serverName*`[:`*portNumber*`]`

IBM DB2 8.1 Type 2 Driver

The JAR files for the DB2 driver are `db2jcc.jar`, `db2jcc_license_cu.jar`, and `db2java.zip`. Set environment variables as follows:

```
LD_LIBRARY_PATH=/usr/db2user/sqllib/lib:${j2ee.home}/lib
DB2DIR=/opt/IBM/db2/V8.1
DB2INSTANCE=db2user
INSTHOME=/usr/db2user
VWSPATH=/usr/db2user/sqllib
THREADS_FLAG=native
```

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `DB2`
- **DataSource Classname:** `com.ibm.db2.jcc.DB2SimpleDataSource`
- **Properties:**
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`databaseName`** - Set as appropriate.
  - **`driverType`** - Set to `2`.
  - **`deferPrepares`** - Set to `false`.

JConnect Type 4 Driver for Sybase ASE 12.5 Databases

The JAR file for the Sybase driver is `jconn2.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Sybase`
- **DataSource Classname:** Specify one of the following:

  ```
  com.sybase.jdbc2.jdbc.SybDataSource
  com.sybase.jdbc2.jdbc.SybXADataSource
  ```

- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.

- **`password`** - Set as appropriate.
- **`databaseName`** - Set as appropriate. Do not specify the complete URL, only the database name.
- **`BE_AS_JDBC_COMPLIANT_AS_POSSIBLE`** - Set to `true`.
- **`FAKE_METADATA`** - Set to `true`.

MM MySQL Type 4 Driver (Non-XA)

The JAR file for the MySQL driver is `mysql-connector-java-`*version*`-bin-g.jar`, for example, `mysql-connector-java-3.1.12-bin-g.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `mysql`
- **DataSource Classname:** Specify one of the following:

  ```
  com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  ```

- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`port`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`databaseName`** - Set as appropriate.
  - **`URL`** - If you are using global transactions, you can set this property instead of `serverName`, `port`, and `databaseName`.

    The MM MySQL Type 4 driver doesn't provide a method to set the required `relaxAutoCommit` property, so you must set it indirectly by setting the **`URL`** property:

    ```
    jdbc:mysql://host:port/database?relaxAutoCommit="true"
    ```

MM MySQL Type 4 Driver (XA Only)

The JAR file for the MySQL driver is `mysql-connector-java-`*version*`-bin-g.jar`, for example, `mysql-connector-java-3.1.12-bin-g.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.

- **Database Vendor:** `mysql`
- **DataSource Classname:** Specify one of the following:

  ```
  com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
  ```

- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`port`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`databaseName`** - Set as appropriate.
  - **`URL`** - If you are using global transactions, you can set this property instead of `serverName`, `port`, and `databaseName`.

    The MM MySQL Type 4 driver doesn't provide a method to set the required `relaxAutoCommit` property, so you must set it indirectly by setting the **`URL`** property:

    ```
    jdbc:mysql://host:port/database?relaxAutoCommit="true"
    ```

Inet Oraxo JDBC Driver for Oracle 8i, 9i, and 10g Databases

The JAR file for the Inet Oracle driver is `Oranxo.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Oracle`
- **DataSource Classname:** `com.inet.ora.OraDataSource`
- **Properties:**
  - **`user`** - Specify the database user.
  - **`password`** - Specify the database password.
  - **`serviceName`** - Specify the URL of the database. The syntax is as follows:

    ```
    jdbc:inetora:server:port:dbname
    ```

    For example:

    ```
    jdbc:inetora:localhost:1521:payrolldb
    ```

In this example,`localhost` is the host name of the machine running the Oracle server, `1521` is the Oracle server's port number, and `payrolldb` is the SID of the database. For more information about the syntax of the database URL, see the Oracle documentation.

- **`serverName`** - Specify the host name or IP address of the database server.
- **`port`** - Specify the port number of the database server.
- **`streamstolob`** - If the size of BLOB or CLOB data types exceeds 4 KB and this driver is used for CMP, this property must be set to `true`.
- **`xa-driver-does-not-support-non-tx-operations`** - Set to the value `true`. Optional: only needed if both non-XA and XA connections are retrieved from the same connection pool. Might degrade performance.

  As an alternative to setting this property, you can create two connection pools, one for non-XA connections and one for XA connections.

Inet Merlia JDBC Driver for Microsoft SQL Server Databases

The JAR file for the Inet Microsoft SQL Server driver is `Merlia.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `mssql`
- **DataSource Classname:** `com.inet.tds.TdsDataSource`
- **Properties:**
  - **`serverName`** - Specify the host name or IP address and the port of the database server.
  - **`port`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.

Inet Sybelux JDBC Driver for Sybase Databases

The JAR file for the Inet Sybase driver is `Sybelux.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Sybase`
- **DataSource Classname:** `com.inet.syb.SybDataSource`

- **Properties:**
  - **`serverName`** - Specify the host name or IP address of the database server.
  - **`portNumber`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`databaseName`** - Set as appropriate. Do not specify the complete URL, only the database name.

Oracle Thin Type 4 Driver for Oracle 8i, 9i, and 10g Databases

The JAR file for the Oracle driver is `ojdbc14.jar`. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Oracle`
- **DataSource Classname:** Specify one of the following:

  ```
  oracle.jdbc.pool.OracleDataSource
  oracle.jdbc.xa.client.OracleXADataSource
  ```

- **Properties:**
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`URL`** - Specify the complete database URL using the following syntax:

    ```
    jdbc:oracle:thin:[user/password]@host[:port]/service
    ```

    For example:

    ```
    jdbc:oracle:thin:@localhost:1521:customer_db
    ```

  - **`xa-driver-does-not-support-non-tx-operations`** - Set to the value `true`. Optional: only needed if both non-XA and XA connections are retrieved from the same connection pool. Might degrade performance.

    As an alternative to setting this property, you can create two connection pools, one for non-XA connections and one for XA connections.

Note – You must set the `oracle-xa-recovery-workaround` property in the Transaction Service for recovery of global transactions to work correctly. For details, see "Transaction Scope" on page 236.

When using this driver, it is not possible to insert more than 2000 bytes of data into a column. To circumvent this problem, use the OCI driver (JDBC type 2).

OCI Oracle Type 2 Driver for Oracle 8i, 9i, and 10g Databases

The JAR file for the OCI Oracle driver is `ojdbc14.jar`. Make sure that the shared library is available through LD_LIBRARY_PATH and that the ORACLE_HOME property is set. Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Oracle`
- **DataSource Classname:** Specify one of the following:

  ```
  oracle.jdbc.pool.OracleDataSource
  oracle.jdbc.xa.client.OracleXADataSource
  ```

- **Properties:**
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`URL`** - Specify the complete database URL using the following syntax:

    ```
    jdbc:oracle:oci:[user/password]@host[:port]/service
    ```

    For example:

    ```
    jdbc:oracle:oci:@localhost:1521:customer_db
    ```

  - **`xa-driver-does-not-support-non-tx-operations`** - Set to the value `true`. Optional: only needed if both non-XA and XA connections are retrieved from the same connection pool. Might degrade performance.

    As an alternative to setting this property, you can create two connection pools, one for non-XA connections and one for XA connections.

IBM Informix Type 4 Driver

Configure the connection pool using the following settings:

- **Name:** Use this name when you configure the JDBC resource later.
- **Resource Type:** Specify the appropriate value.
- **Database Vendor:** `Informix`
- **DataSource Classname:** Specify one of the following:

  ```
  com.informix.jdbcx.IfxDataSource
  com.informix.jdbcx.IfxXADataSource
  ```

- **Properties:**
  - **`serverName`** - Specify the Informix database server name.
  - **`portNumber`** - Specify the port number of the database server.
  - **`user`** - Set as appropriate.
  - **`password`** - Set as appropriate.
  - **`databaseName`** - Set as appropriate. This is optional.
  - **`IfxIFXHost`** - Specify the host name or IP address of the database server.

◆ ◆ ◆ CHAPTER 12

Using the Transaction Service

The J2EE platform provides several abstractions that simplify development of dependable transaction processing for applications. This chapter discusses J2EE transactions and transaction support in the Sun Java System Application Server.

This chapter contains the following sections:

For more information about the Java™ Transaction API (JTA) and Java Transaction Service (JTS), see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* and the following sites: `http://java.sun.com/products/jta/` and `http://java.sun.com/products/jts/`.

You might also want to read the chapter on transactions in the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`.

Transaction Resource Managers

There are three types of transaction resource managers:

- Databases - Use of transactions prevents databases from being left in inconsistent states due to incomplete updates. For information about JDBC transaction isolation levels, see "Using JDBC Transaction Isolation Levels" on page 223.

  The Application Server supports a variety of JDBC™ XA drivers. For a list of the JDBC drivers currently supported by the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Release Notes*. For configurations of supported and other drivers, see "Configurations for Specific JDBC Drivers" on page 224.

- Java Message Service (JMS) Providers - Use of transactions ensures that messages are reliably delivered. The Application Server is integrated with Sun Java System Message Queue, a fully capable JMS provider. For more information about transactions and the JMS API, see Chapter 14, "Using the Java Message Service."
- J2EE™ Connector Architecture (CA) components - Use of transactions prevents legacy EIS systems from being left in inconsistent states due to incomplete updates. For more information about connectors, see Chapter 9, "Developing Connectors."

For details about how transaction resource managers, the transaction service, and applications interact, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Note – In the Application Server, the transaction manager is a privileged interface. However, applications can access `UserTransaction`. For more information, see "Naming Environment for J2EE Application Components" on page 240.

Transaction Scope

A *local* transaction involves only one non-XA resource and requires that all participating application components execute within one process. Local transaction optimization is specific to the resource manager and is transparent to the J2EE application.

In the Application Server, a JDBC resource is non-XA if it meets any of the following criteria:

- In the JDBC connection pool configuration, the DataSource class does not implement the `javax.sql.XADataSource` interface.
- The Global Transaction Support box is not checked, or the Resource Type setting does not exist or is not set to `javax.sql.XADataSource`.

A transaction remains local if the following conditions remain true:

- One and only one non-XA resource is used. If any additional non-XA resource is used, the transaction is aborted.
- No transaction importing or exporting occurs.

Transactions that involve multiple resources or multiple participant processes are *distributed* or *global* transactions. A global transaction can involve one non-XA resource if last agent optimization is enabled. Otherwise, all resourced must be XA. The `use-last-agent-optimization` property is set to `true` by default. For details about how to set this property, see "Configuring the Transaction Service" on page 237.

If only one XA resource is used in a transaction, one-phase commit occurs, otherwise the transaction is coordinated with a two-phase commit protocol.

A two-phase commit protocol between the transaction manager and all the resources enlisted for a transaction ensures that either all the resource managers commit the transaction or they all

abort. When the application requests the commitment of a transaction, the transaction manager issues a `PREPARE_TO_COMMIT` request to all the resource managers involved. Each of these resources can in turn send a reply indicating whether it is ready for commit (`PREPARED`) or not (`NO`). Only when all the resource managers are ready for a commit does the transaction manager issue a commit request (`COMMIT`) to all the resource managers. Otherwise, the transaction manager issues a rollback request (`ABORT`) and the transaction is rolled back.

The Application Server provides workarounds for some known issues with the recovery implementations of the following JDBC drivers. These workarounds are used unless explicitly disabled.

- Oracle thin driver - The `XAResource.recover` method repeatedly returns the same set of in-doubt Xids regardless of the input flag. According to the XA specifications, the Transaction Manager initially calls this method with TMSTARTSCAN and then with TMNOFLAGS repeatedly until no Xids are returned. The `XAResource.commit` method also has some issues.

  To disable the Application Server workaround, set the `oracle-xa-recovery-workaround` property value to `false`. For details about how to set this property, see "Configuring the Transaction Service" on page 237.

Note – These workarounds do not imply support for any particular JDBC driver.

Configuring the Transaction Service

You can configure the transaction service in the Application Server in the following ways:

- To configure the transaction service using the Administration Console, open the Transaction Service component under the relevant configuration. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- To configure the transaction service, use the `asadmin set` command to set the following attributes:

```
server.transaction-service.automatic-recovery = false
server.transaction-service.heuristic-decision = rollback
server.transaction-service.keypoint-interval = 2048
server.transaction-service.retry-timeout-in-seconds = 600
server.transaction-service.timeout-in-seconds = 0
server.transaction-service.tx-log-dir = domain-dir/logs
```

  You can also set these properties:

```
server.transaction-service.property.oracle-xa-recovery-workaround = false
server.transaction-service.property.disable-distributed-transaction-logging = false
server.transaction-service.property.xaresource-txn-timeout = 600
```

```
server.transaction-service.property.pending-txn-cleanup-interval = 60
server.transaction-service.property.use-last-agent-optimization = true
```

You can use the `asadmin get` command to list all the transaction service attributes and properties. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Transaction Logging

The transaction service writes transactional activity into transaction logs so that transactions can be recovered. You can control transaction logging in these ways:

- Set the location of the transaction log files using the Transaction Log Location setting in the Administration Console, or set the `tx-log-dir` attribute using the `asadmin set` command.
- Turn off transaction logging by setting the `disable-distributed-transaction-logging` property to `true`. Do this *only* if performance is more important than transaction recovery.

◆ ◆ ◆ CHAPTER 13

Using the Java Naming and Directory Interface

A *naming service* maintains a set of bindings, which relate names to objects. The J2EE™ naming service is based on the Java Naming and Directory Interface™ (JNDI) API. The JNDI API allows application components and clients to look up distributed resources, services, and EJB™ components. For general information about the JNDI API, see `http://java.sun.com/products/jndi/`.

You can also see the JNDI tutorial at `http://java.sun.com/products/jndi/tutorial/`.

This chapter contains the following sections:

Accessing the Naming Context

The Application Server provides a naming environment, or *context*, which is compliant with standard J2EE 1.4 requirements. A `Context` object provides the methods for binding names to objects, unbinding names from objects, renaming objects, and listing the bindings. The `InitialContext` is the handle to the J2EE naming service that application components and clients use for lookups.

The JNDI API also provides subcontext functionality. Much like a directory in a file system, a subcontext is a context within a context. This hierarchical structure permits better organization of information. For naming services that support subcontexts, the `Context` class also provides methods for creating and destroying subcontexts.

The rest of this section covers these topics:

- "Naming Environment for Lifecycle Modules" on page 242

Note – Each resource within a server instance must have a unique name. However, two resources in different server instances or different domains can have the same name.

Naming Environment for J2EE Application Components

The namespace for objects looked up in a J2EE environment is organized into different subcontexts, with the standard prefix `java:comp/env`.

The following table describes standard JNDI subcontexts for connection factories in the Application Server.

TABLE 13–1 Standard JNDI Subcontexts for Connection Factories

Resource Manager	Connection Factory Type	JNDI Subcontext
JDBC™	`javax.sql.DataSource`	`java:comp/env/jdbc`
Transaction Service	`javax.transaction.UserTransaction`	`java:comp/UserTransaction`
JMS	`javax.jms.TopicConnectionFactory` `javax.jms.QueueConnectionFactory`	`java:comp/env/jms`
JavaMail™	`javax.mail.Session`	`java:comp/env/mail`
URL	`java.net.URL`	`java:comp/env/url`
Connector	`javax.resource.cci.ConnectionFactory`	`java:comp/env/eis`

Accessing EJB Components Using the CosNaming Naming Context

The preferred way of accessing the naming service, even in code that runs outside of a J2EE container, is to use the no-argument `InitialContext` constructor. However, if EJB client code explicitly instantiates an `InitialContext` that points to the CosNaming naming service, it is necessary to set these properties in the client JVM when accessing EJB components:

```
-Djavax.rmi.CORBA.UtilClass=com.sun.corba.ee.impl.javax.rmi.CORBA.Util
-Dorg.omg.CORBA.ORBClass=com.sun.corba.ee.impl.orb.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.sun.corba.ee.impl.orb.ORBSingleton
-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
```

Accessing EJB Components in a Remote Application Server

The recommended approach for looking up an EJB component in a remote Application Server from a client that is a servlet or EJB component is to use the Interoperable Naming Service syntax. Host and port information is prepended to any global JNDI names and is automatically resolved during the lookup. The syntax for an interoperable global name is as follows:

`corbaname:iiop:`*host*`:`*port*`#a/b/`*name*

This makes the programming model for accessing EJB components in another Application Server exactly the same as accessing them in the same server. The deployer can change the way the EJB components are physically distributed without having to change the code.

For J2EE components, the code still performs a `java:comp/env` lookup on an EJB reference. The only difference is that the deployer maps the `ejb-reference` element to an interoperable name in an Application Server deployment descriptor file instead of a simple global JNDI name.

For example, suppose a servlet looks up an EJB reference using `java:comp/env/ejb/Foo`, and the target EJB component has a global JNDI name of `a/b/Foo`.

The `ejb-ref` element in `sun-web.xml` looks like this:

```
<ejb-ref>
   <ejb-ref-name>ejb/Foo</ejb-ref-name>
   <jndi-name>corbaname:iiop:host:port#a/b/Foo</jndi-name>
<ejb-ref>
```

The code looks like this:

```
Context ic = new InitialContext();
Object o = ic.lookup("java:comp/env/ejb/Foo");
```

For a client that doesn't run within a J2EE container, the code just uses the interoperable global name instead of the simple global JNDI name. For example:

```
Context ic = new InitialContext();
Object o = ic.lookup("corbaname:iiop:host:port#a/b/Foo");
```

Objects stored in the interoperable naming context and component-specific (`java:comp/env`) naming contexts are transient. On each server startup or application reloading, all relevant objects are re-bound to the namespace.

Naming Environment for Lifecycle Modules

Lifecycle listener modules provide a means of running short or long duration Java-based tasks within the application server environment, such as instantiation of singletons or RMI servers. These modules are automatically initiated at server startup and are notified at various phases of the server life cycle. For details about lifecycle modules, see Chapter 10, "Developing Lifecycle Listeners."

The configured properties for a lifecycle module are passed as properties during server initialization (the `INIT_EVENT`). The initial JNDI naming context is not available until server initialization is complete. A lifecycle module can get the `InitialContext` for lookups using the method `LifecycleEventContext.getInitialContext()` during, and only during, the `STARTUP_EVENT`, `READY_EVENT`, or `SHUTDOWN_EVENT` server life cycle events.

Configuring Resources

The Application Server exposes the following special resources in the naming environment. Full administration details are provided in the following sections:

- "External JNDI Resources" on page 242
- "Custom Resources" on page 242

External JNDI Resources

An external JNDI resource defines custom JNDI contexts and implements the `javax.naming.spi.InitialContextFactory` interface. There is no specific JNDI parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create an external JNDI resource in one of these ways:

- To create an external JNDI resource using the Administration Console, open the Resources component, open the JNDI component, and select External Resources. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- To create an external JNDI resource, use the `asadmin create-jndi-resource` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Custom Resources

A custom resource specifies a custom server-wide resource object factory that implements the `javax.naming.spi.ObjectFactory` interface. There is no specific JNDI parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create a custom resource in one of these ways:

- To create a custom resource using the Administration Console, open the Resources component, open the JNDI component, and select Custom Resources. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- To create a custom resource, use the `asadmin create-custom-resource` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Mapping References

The following XML elements map JNDI names configured in the Application Server to resource references in application client, EJB, and web application components:

- `resource-env-ref` - Maps the `resource-env-ref` element in the corresponding J2EE XML file to the absolute JNDI name configured in the Application Server.
- `resource-ref` - Maps the `resource-ref` element in the corresponding J2EE XML file to the absolute JNDI name configured in the Application Server.
- `ejb-ref` - Maps the `ejb-ref` element in the corresponding J2EE XML file to the absolute JNDI name configured in the Application Server.

JNDI names for EJB components must be unique. For example, appending the application name and the module name to the EJB name is one way to guarantee unique names. In this case, `mycompany.pkging.pkgingEJB.MyEJB` would be the JNDI name for an EJB in the module `pkgingEJB.jar`, which is packaged in the `pkging.ear` application.

These elements are part of the `sun-web-app.xml`, `sun-ejb-ref.xml`, and `sun-application-client.xml` deployment descriptor files. For more information about how these elements behave in each of the deployment descriptor files, see Appendix A, "Deployment Descriptor Files."

The rest of this section uses an example of a JDBC resource lookup to describe how to reference resource factories. The same principle is applicable to all resources (such as JMS destinations, JavaMail sessions, and so on).

The `resource-ref` element in the `sun-web-app.xml` deployment descriptor file maps the JNDI name of a resource reference to the `resource-ref` element in the `web-app.xml` J2EE deployment descriptor file.

The resource lookup in the application code looks like this:

```
InitialContext ic = new InitialContext();
String dsName = "java:comp/env/jdbc/HelloDbDs";
DataSource ds = (javax.sql.DataSource)ic.lookup(dsName);
Connection connection = ds.getConnection();
```

The resource being queried is listed in the `res-ref-name` element of the `web.xml` file as follows:

```
<resource-ref>
   <description>DataSource Reference</description>
   <res-ref-name>jdbc/HelloDbDs</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

The `resource-ref` section in a Application Server specific deployment descriptor, for example `sun-web.xml`, maps the `res-ref-name` (the name being queried in the application code) to the JNDI name of the JDBC resource. The JNDI name is the same as the name of the JDBC resource as defined in the resource file when the resource is created.

```
<resource-ref>
   <res-ref-name>jdbc/HelloDbDs</res-ref-name>
   <jndi-name>jdbc/HelloDbDataSource</jndi-name>
</resource-ref>
```

The JNDI name in the Application Server specific deployment descriptor must match the JNDI name you assigned to the resource when you created and configured it.

◆ ◆ ◆ CHAPTER 14

Using the Java Message Service

This chapter describes how to use the Java™ Message Service (JMS) API. The Sun Java System Application Server has a fully integrated JMS provider: the Sun Java System Message Queue software.

For general information about the JMS API, see the *J2EE 1.4 Tutorial* at `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JMS.html#wp84181`.

For detailed information about JMS concepts and JMS support in the Application Server, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

This chapter contains the following sections:

The JMS Provider

The Application Server support for JMS messaging, in general, and for message-driven beans, in particular, requires messaging middleware that implements the JMS specification: a JMS provider. The Application Server uses the Sun Java System Message Queue software as its native JMS provider. The Message Queue software is tightly integrated into theApplication Server, providing transparent JMS messaging support. This support is known within Application Server as the *JMS Service*. The JMS Service requires only minimal administration.

The relationship of the Message Queue software to the Application Server can be one of these types: `LOCAL` or `REMOTE`. The results of these choices are as follows:

- If the type is `LOCAL`, the Message Queue broker starts when the Application Server starts. This is the default.
- If the type is `REMOTE`, the Message Queue broker must be started separately. For information about starting the broker, see the *Sun Java System Message Queue 3.7 UR1 Administration Guide*.

For more information about setting the type and the default JMS host, see "Configuring the JMS Service" on page 247.

For more information about the Message Queue software, refer to the documentation at `http://docs.sun.com/app/docs/coll/1307.2`.

For general information about the JMS API, see the JMS web page at `http://java.sun.com/products/jms/index.html`.

Message Queue Resource Adapter

The Sun Java System Message Queue software is integrated into the Application Server using a resource adapter that is compliant with the Connector 1.5 specification. The module name of this system resource adapter is `jmsra`. Every JMS resource is converted to a corresponding connector resource of this resource adapter as follows:

- **Connection Factory:** A connector connection pool with a `max-pool-size` of `250` and a corresponding connector resource.
- **Destination (Topic or Queue):** A connector administered object.

You use connector configuration tools to manage JMS resources. For more information, see Chapter 9, "Developing Connectors."

Administration of the JMS Service

To configure the JMS Service and prepare JMS resources for use in applications deployed to the Application Server, you must perform these tasks:

For more information about JMS administration tasks, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* and the *Sun Java System Message Queue 3.7 UR1 Administration Guide*.

Configuring the JMS Service

The JMS Service configuration is available to all inbound and outbound connections pertaining to the Application Server. You can edit the JMS Service configuration in the following ways:

- To edit the JMS Service configuration using the Administration Console, open the Java Message Service component under the relevant configuration. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- To configure the JMS service, use the `asadmin set` command to set the following attributes:

  ```
  server.jms-service.init-timeout-in-seconds = 60
  server.jms-service.type = LOCAL
  server.jms-service.start-args =
  server.jms-service.default-jms-host = default_JMS_host
  server.jms-service.reconnect-interval-in-seconds = 60
  server.jms-service.reconnect-attempts = 3
  server.jms-service.reconnect-enabled = true
  server.jms-service.addresslist-behavior = random
  server.jms-service.addresslist-iterations = 3
  server.jms-service.mq-scheme = mq
  server.jms-service.mq-service = jms
  ```

  You can also set these properties:

  ```
  server.jms-service.property.instance-name = imqbroker
  server.jms-service.property.instance-name-suffix =
  server.jms-service.property.append-version = false
  ```

  You can use the `asadmin get` command to list all the JMS service attributes and properties. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

You can override the JMS Service configuration using JMS connection factory settings. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

Note – The Application Server must be restarted after configuration of the JMS Service.

The Default JMS Host

A JMS host refers to a Sun Java System Message Queue broker. A default JMS host for the JMS service is provided, named `default_JMS_host`. This is the JMS host that the Application Server instance starts when the JMS Service type is configured as `LOCAL`.

If you have created a multi-broker cluster in the Message Queue software, delete the default JMS host, then add the Message Queue cluster's brokers as JMS hosts. In this case, the default JMS host becomes the first JMS host in the `AddressList`. (For more information about the `AddressList`, see "JMS Connection Features" on page 250. You can also explicitly set the default JMS host; see "Configuring the JMS Service" on page 247.

When the Application Server uses a Message Queue cluster, it executes Message Queue specific commands on the default JMS host. For example, when a physical destination is created for a Message Queue cluster of three brokers, the command to create the physical destination is executed on the default JMS host, but the physical destination is used by all three brokers in the cluster.

Creating JMS Hosts

You can create additional JMS hosts in the following ways:

- Use the Administration Console. Open the Java Message Service component under the relevant configuration, then select the JMS Hosts component. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-jms-host` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Checking Whether the JMS Provider Is Running

You can use the `asadmin jms-ping` command to check whether a Sun Java System Message Queue instance is running. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

Creating Physical Destinations

Produced messages are delivered for routing and subsequent delivery to consumers using *physical destinations* in the JMS provider. A physical destination is identified and encapsulated by an administered object (a `Topic` or `Queue` destination resource) that an application component uses to specify the destination of messages it is producing and the source of messages it is consuming.

If a message-driven bean is deployed and the physical destination it listens to doesn't exist, the Application Server automatically creates the physical destination. However, it is good practice to create the physical destination beforehand.

You can create a JMS physical destination in the following ways:

- Use the Administration Console. Open the Resources component, open the JMS Resources component, then select Physical Destinations. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-jmsdest` command. This command acts on the default JMS host. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

To create a destination resource, see "Creating JMS Resources: Destinations and Connection Factories" on page 249.

Creating JMS Resources: Destinations and Connection Factories

You can create two kinds of JMS resources in the Application Server:

- **Connection Factories:** administered objects that implement the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory` interfaces.
- **Destination Resources:** administered objects that implement the `Queue` or `Topic` interfaces.

In either case, the steps for creating a JMS resource are the same. You can create a JMS resource in the following ways:

- To create a JMS resource using the Administration Console, open the Resources component, then open the JMS Resources component. Click Connection Factories to create a connection factory, or click Destination Resources to create a queue or topic. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- A JMS resource is a type of connector. To create a JMS resource using the command line, see "Deploying and Configuring a Stand-Alone Connector Module" on page 203.

Note – All JMS resource properties that used to work with version 7 of the Application Server are supported for backward compatibility.

Restarting the JMS Client After JMS Configuration

When a JMS client accesses a JMS administered object for the first time, the client JVM retrieves the JMS service configuration from the Application Server. Further changes to the configuration are not available to the client JVM until the client is restarted.

JMS Connection Features

The Sun Java System Message Queue software supports the following JMS connection features:

- "Connection Pooling" on page 250
- "Connection Failover" on page 251

Both these features use the `AddressList` configuration, which is populated with the hosts and ports of the JMS hosts defined in the Application Server. The `AddressList` is updated whenever a JMS host configuration changes. The `AddressList` is inherited by any JMS resource when it is created and by any MDB when it is deployed.

Note – In the Sun Java System Message Queue software, the `AddressList` property is called `imqAddressList`.

Connection Pooling

The Application Server pools JMS connections automatically.

To dynamically modify connection pool properties using the Administration Console, go to either the Connection Factories page (see "Creating JMS Resources: Destinations and Connection Factories" on page 249) or the Connector Connection Pools page (see "Deploying and Configuring a Stand-Alone Connector Module" on page 203).

To use the command line, use the `asadmin create-connector-connection-pool` command to manage the pool (see "Deploying and Configuring a Stand-Alone Connector Module" on page 203.

The `addresslist-behavior` JMS service attribute is set to `random` by default. This means that each `ManagedConnection` (physical connection) created from the `ManagedConnectionFactory` selects its primary broker in a random way from the `AddressList`.

When a JMS connection pool is created, there is one `ManagedConnectionFactory` instance associated with it. If you configure the `AddressList` as a `ManagedConnectionFactory` property, the `AddressList` configuration in the `ManagedConnectionFactory` takes precedence over the one defined in the Application Server.

Connection Failover

To specify whether the Application Server tries to reconnect to the primary broker if the connection is lost, set the `reconnect-enabled` attribute in the JMS service. To specify the number of retries and the time between retries, set the `reconnect-attempts` and `reconnect-interval-in-seconds` attributes, respectively.

If reconnection is enabled and the primary broker goes down, the Application Server tries to reconnect to another broker in the `AddressList`. The `AddressList` is updated whenever a JMS host configuration changes. The logic for scanning is decided by two JMS service attributes, `addresslist-behavior` and `addresslist-iterations`.

You can override these settings using JMS connection factory settings. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.

The Sun Java System Message Queue software transparently transfers the load to another broker when the failover occurs. JMS semantics are maintained during failover.

Transactions and Non-Persistent Messages

During transaction recovery, non-persistent messages might be lost. If the broker fails between the transaction manager's prepare and commit operations, any non-persistent message in the transaction is lost and cannot be delivered. A message that is not saved to a persistent store is not available for transaction recovery.

ConnectionFactory Authentication

If your web, EJB, or client module has `res-auth` set to `Container`, but you use the `ConnectionFactory.createConnection("user","password")` method to get a connection, the Application Server searches the container for authentication information before using the supplied user and password. Version 7 of the Application Server threw an exception in this situation.

Message Queue varhome Directory

The Sun Java System Message Queue software uses a default directory for storing data such as persistent messages and its log file. This directory is called `varhome`. The Application Server uses *domain-dir*`/imq` as the `varhome` directory. Thus, for the default Application Server domain, Message Queue data is stored in the following location:

install-dir`/domains/domain1/imq/var/instances/imqbroker`

Version 7 of the Application Server stored this data in the following location:

install-dir`/imq/var/instances/domain1_server`

When executing Message Queue scripts such as *install-dir*`/imq/bin/imqusermgr`, use the `-varhome` option. For example:

```
imqusermgr -varhome $AS_INSTALL/domains/domain1/imq add -u testuser
-p testpassword
```

Delivering SOAP Messages Using the JMS API

Web service clients use the Simple Object Access Protocol (SOAP) to communicate with web services. SOAP uses a combination of XML-based data structuring and Hyper Text Transfer Protocol (HTTP) to define a standardized way of invoking methods in objects distributed in diverse operating environments across the Internet.

For more information about SOAP, see the Apache SOAP web site at `http://xml.apache.org/soap/index.html`.

You can take advantage of the JMS provider's reliable messaging when delivering SOAP messages. You can convert a SOAP message into a JMS message, send the JMS message, then convert the JMS message back into a SOAP message. The following sections explain how to do these conversions:

- "To send SOAP messages using the JMS API" on page 252
- "To receive SOAP messages using the JMS API" on page 253

▼ To send SOAP messages using the JMS API

1 **Import the `MessageTransformer` library.**

```
import com.sun.messaging.xml.MessageTransformer;
```

This is the utility whose methods you use to convert SOAP messages to JMS messages and the reverse. You can then send a JMS message containing a SOAP payload as if it were a normal JMS message.

2 **Initialize the `TopicConnectionFactory`, `TopicConnection`, `TopicSession`, and publisher.**

```
tcf = new TopicConnectionFactory();
tc = tcf.createTopicConnection();
session = tc.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
topic = session.createTopic(topicName);
publisher = session.createPublisher(topic);
```

3 **Construct a SOAP message using the SOAP with Attachments API for Java (SAAJ).**

```
*construct a default soap MessageFactory */
MessageFactory mf = MessageFactory.newInstance();
* Create a SOAP message object.*/
SOAPMessage soapMessage = mf.createMessage();
/** Get SOAP part.*/
SOAPPart soapPart = soapMessage.getSOAPPart();
/* Get SOAP envelope. */
SOAPEnvelope soapEnvelope = soapPart.getEnvelope();
/* Get SOAP body.*/
SOAPBody soapBody = soapEnvelope.getBody();
/* Create a name object. with name space */
/* http://www.sun.com/imq. */
Name name = soapEnvelope.createName("HelloWorld", "hw",
 "http://www.sun.com/imq");
* Add child element with the above name. */
SOAPElement element = soapBody.addChildElement(name)
/* Add another child element.*/
element.addTextNode( "Welcome to Sun Java System Web Services." );
/* Create an atachment with activation API.*/
URL url = new URL ("http://java.sun.com/webservices/");
DataHandler dh = new DataHandler (url);
AttachmentPart ap = soapMessage.createAttachmentPart(dh);
/*set content type/ID. */
ap.setContentType("text/html");
ap.setContentId("cid-001");
/**  add the attachment to the SOAP message.*/
soapMessage.addAttachmentPart(ap);
soapMessage.saveChanges();
```

4 **Convert the SOAP message to a JMS message by calling the `MessageTransformer.SOAPMessageintoJMSMessage()` method.**

```
Message m = MessageTransformer.SOAPMessageIntoJMSMessage (soapMessage,
session );
```

5 **Publish the JMS message.**

```
publisher.publish(m);
```

6 **Close the JMS connection.**

```
tc.close();
```

▼ To receive SOAP messages using the JMS API

1 **Import the `MessageTransformer` library.**

```
import com.sun.messaging.xml.MessageTransformer;
```

This is the utility whose methods you use to convert SOAP messages to JMS messages and the reverse. The JMS message containing the SOAP payload is received as if it were a normal JMS message.

2 **Initialize the `TopicConnectionFactory`, `TopicConnection`, `TopicSession`, `TopicSubscriber`, and `Topic`.**

```
messageFactory = MessageFactory.newInstance();
tcf = new com.sun.messaging.TopicConnectionFactory();
tc = tcf.createTopicConnection();
session = tc.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
topic = session.createTopic(topicName);
subscriber = session.createSubscriber(topic);
subscriber.setMessageListener(this);
tc.start();
```

3 **Use the `OnMessage` method to receive the message. Use the `SOAPMessageFromJMSMessage` method to convert the JMS message to a SOAP message.**

```
public void onMessage (Message message) {
SOAPMessage soapMessage =
 MessageTransformer.SOAPMessageFromJMSMessage( message,
 messageFactory ); }
```

4 **Retrieve the content of the SOAP message.**

◆ ◆ ◆ CHAPTER 15

Using the JavaMail API

This chapter describes how to use the JavaMail™ API, which provides a set of abstract classes defining objects that comprise a mail system.

This chapter contains the following sections:

Introducing JavaMail

The JavaMail API defines classes such as `Message`, `Store`, and `Transport`. The API can be extended and can be subclassed to provide new protocols and to add functionality when necessary. In addition, the API provides concrete subclasses of the abstract classes. These subclasses, including `MimeMessage` and `MimeBodyPart`, implement widely used Internet mail protocols and conform to the RFC822 and RFC2045 specifications. The JavaMail API includes support for the IMAP4, POP3, and SMTP protocols.

The JavaMail architectural components are as follows:

- The *abstract layer* declares classes, interfaces, and abstract methods intended to support mail handling functions that all mail systems support.
- The *internet implementation layer* implements part of the abstract layer using the RFC822 and MIME internet standards.
- JavaMail uses the *JavaBeans Activation Framework* (JAF) to encapsulate message data and to handle commands intended to interact with that data.

For more information, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* and the JavaMail specification at `http://java.sun.com/products/javamail/`.

Creating a JavaMail Session

You can create a JavaMail session in the following ways:

- In the Administration Console, open the Resources component and select JavaMail Sessions. For details, see the *Sun Java System Application Server Platform Edition 8.2 Administration Guide*.
- Use the `asadmin create-javamail-resource` command. For details, see the *Sun Java System Application Server Platform Edition 8.2 Reference Manual*.

JavaMail Session Properties

You can set properties for a JavaMail `Session` object. Every property name must start with a `mail-` prefix. The Application Server changes the dash (`-`) character to a period (`.`) in the name of the property and saves the property to the `MailConfiguration` and JavaMail `Session` objects. If the name of the property doesn't start with `mail-`, the property is ignored.

For example, if you want to define the property `mail.from` in a JavaMail `Session` object, first define the property as follows:

- Name - `mail-from`
- Value - `john.doe@sun.com`

After you get the JavaMail `Session` object, you can get the `mail.from` property to retrieve the value as follows:

```
String password = session.getProperty("mail.from");
```

Looking Up a JavaMail Session

The standard Java Naming and Directory Interface™ (JNDI) subcontext for JavaMail sessions is `java:comp/env/mail`.

Registering JavaMail sessions in the `mail` naming subcontext of a JNDI namespace, or in one of its child subcontexts, is standard. The JNDI namespace is hierarchical, like a file system's directory structure, so it is easy to find and nest references. A JavaMail session is bound to a logical JNDI name. The name identifies a subcontext, `mail`, of the root context, and a logical name. To change the JavaMail session, you can change its entry in the JNDI namespace without having to modify the application.

The resource lookup in the application code looks like this:

```
InitialContext ic = new InitialContext();
String snName = "java:comp/env/mail/MyMailSession";
Session session = (Session)ic.lookup(snName);
```

For more information about the JNDI API, see Chapter 13, "Using the Java Naming and Directory Interface."

Sending and Reading Messages Using JavaMail

▼ To send a message using JavaMail

1 Import the packages that you need.

```
import java.util.*;
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.naming.*;
```

2 Look up the JavaMail session.

```
InitialContext ic = new InitialContext();
String snName = "java:comp/env/mail/MyMailSession";
Session session = (Session)ic.lookup(snName);
```

For more information, see "Looking Up a JavaMail Session" on page 256.

3 Override the JavaMail session properties if necessary.

For example:

```
Properties props = session.getProperties();
props.put("mail.from", "user2@mailserver.com");
```

4 Create a `MimeMessage`.

The `msgRecipient`, `msgSubject`, and `msgTxt` variables in the following example contain input from the user:

```
Message msg = new MimeMessage(session);
msg.setSubject(msgSubject);
msg.setSentDate(new Date());
msg.setFrom();
msg.setRecipients(Message.RecipientType.TO,
   InternetAddress.parse(msgRecipient, false));
msg.setText(msgTxt);
```

5 **Send the message.**

```
Transport.send(msg);
```

▼ To read a message using JavaMail

1 **Import the packages that you need.**

```
import java.util.*;
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.naming.*;
```

2 **Look up the JavaMail session.**

```
InitialContext ic = new InitialContext();
String snName = "java:comp/env/mail/MyMailSession";
Session session = (javax.mail.Session)ic.lookup(snName);
```

For more information, see "Looking Up a JavaMail Session" on page 256.

3 **Override the JavaMail session properties if necessary.**

For example:

```
Properties props = session.getProperties();
props.put("mail.from", "user2@mailserver.com");
```

4 **Get a `Store` object from the `Session`, then connect to the mail server using the Store object's `connect()` method.**

You must supply a mail server name, a mail user name, and a password.

```
Store store = session.getStore();
store.connect("MailServer", "MailUser", "secret");
```

5 **Get the INBOX folder.**

```
Folder folder = store.getFolder("INBOX");
```

6 **It is efficient to read the `Message` objects (which represent messages on the server) into an array.**

```
Message[] messages = folder.getMessages();
```

◆ ◆ ◆ CHAPTER 16

Using the Java Management Extensions (JMX) API

The Sun Java™ System Application Server uses Java Management Extensions (JMX™) technology for monitoring, management and notification purposes. Management and monitoring of the Application Server is performed by the Application Server Management Extensions (AMX), which exposes managed resources for remote management via the JMX Application Programming Interface (API).

The Application Server incorporates the JMX 1.2 Reference Implementation, that was developed by the Java Community Process as Java Specification Request (JSR) 3, and the JMX Remote API 1.0 Reference Implementation (JSR 160).

This chapter assumes some familiarity with the JMX technology, but the AMX interfaces can be used for the most part without understanding JMX.

The JMX specifications and Reference Implementations are available for download at `http://java.sun.com/products/JavaManagement/download.html`.

This chapter contains the following topics:

About AMX

This section describes the **A**pplication Server **M**anagement e**X**tensions (AMX). AMX is an API that exposes all of the Application Server configuration and monitoring MBeans as easy-to-use client-side dynamic proxies implementing the AMX interfaces.

Full API documentation for the AMX API is provided in the following Application Server package:

```
com.sun.appserv.management
```

The Application Server is based around the concept of *administration domains*, which consist of one or more *managed resources*. A managed resource can be an Application Server or a manageable entity within a server. A managed resource is of a particular type, and each resource type exposes a set of attributes and administrative operations that change the resource's state.

Managed resources are exposed as JMX *management beans*, or *MBeans*. While the MBeans can be accessed via standard JMX APIs (for example, `MBeanServerConnection`), most users find the use of the AMX client-side dynamic proxies much more convenient.

All the vital components of the Application Server are visible for monitoring and management via AMX. You can use third-party tools to perform all common administrative tasks programmatically, based on the JMX and JMX Remote API standards.

The AMX API consists of a set of proxy interfaces. MBeans are registered in the JMX runtime contained in the Domain Administration Server (DAS). AMX provides routines to obtain proxies for MBeans, starting with a root-level domain MBean.

You can navigate generically through the MBean hierarchy using the `com.sun.appserv.management.base.Container` interface. When using AMX, the interfaces defined are implemented by client-side dynamic proxies, but they also implicitly define the `MBeanInfo` that is made available by the MBean or MBeans corresponding to it. Certain operations defined in the interface might have a different return type or a slightly different name when accessed through the MBean directly. This results from the fact that direct access to JMX requires the use of `ObjectName`, whereas use of the AMX interfaces is via strongly typed proxies implementing the interface(s).

AMX MBeans

All AMX MBeans are represented as interfaces in a subpackage of `com.sun.appserv.management` and are implemented by dynamic proxies on the client-side. While you can access AMX MBeans directly through standard JMX APIs, most users find the use of AMX interface (proxy) classes to be most convenient.

An AMX MBean belongs to an application server domain. There is exactly one domain per DAS. Thus all MBeans accessible through the DAS belong to a single Application Server

administrative domain. All MBeans in an Application Server administrative domain, and hence within the DAS, belong to the JMX domain `amx`. Any MBeans that do not have the JMX domain `amx` are not part of AMX, and are neither documented nor supported for use by clients. All AMX MBeans can be reached navigationally through the `DomainRoot`.

AMX defines different types of MBean, namely, *configuration* MBeans, *monitoring* MBeans, *utility* MBeans and *J2EE management (JSR 77)* MBeans. These MBeans are logically related in the following ways:

- They all implement the `com.sun.appserv.management.base.AMX` interface.
- They all have a `j2eeType` and `name` property within their `ObjectName` (see `com.sun.appserv.management.base.XTypes` and `com.sun.appserv.management.j2ee.J2EETypes` for the available values of the `j2eeType` property).
- All MBeans that logically contain other MBeans implement the `com.sun.appserv.management.base.Container` interface.
- JSR 77 MBeans that have a corresponding configuration or monitoring peer expose it via `getConfigPeer()` or `getMonitoringPeer()`. However, there are many configuration and monitoring MBeans that do not correspond to JSR 77 MBeans.

Configuration MBeans

Configuration information for a given Application Server domain is stored in a central repository that is shared by all instances in that domain. The central repository can only be written to by the DAS. However, configuration information in the central repository is made available to administration clients via AMX MBeans.

The configuration MBeans are those that modify the underlying `domain.xml` or related files. Collectively, they form a model representing the configuration and deployment repository and the operations that can be performed on them.

The `Group` Attribute of configuration MBeans, obtained from `getGroup()`, has a value of `com.sun.appserv.management.base.AMX.GROUP_CONFIGURATION`.

Monitoring MBeans

Monitoring MBeans provide transient monitoring information about all the vital components of the Application Server.

The `Group` Attribute of monitoring MBeans, obtained from `getGroup()`, has a value of `com.sun.appserv.management.base.AMX.GROUP_MONITORING`.

Utility MBeans

Utility MBeans provide commonly used services to the Application Server.

The `Group` Attribute of utility MBeans, obtained from `getGroup()`, has a value of `com.sun.appserv.management.base.AMX.GROUP_UTILITY`.

J2EE Management MBeans

The J2EE management MBeans implement, and in some cases extend, the management hierarchy as defined by JSR 77, which specifies the management model for the whole J2EE platform. One of the management APIs implemented in JSR 77 is the JMX API.

The implementation of JSR 77 in AMX offers access to and monitoring of MBeans via J2EE management MBeans, by using the `getMonitoringPeer()` and `getConfigPeer()` methods.

The J2EE management MBeans can be thought of as the central hub from which other MBeans are obtained.

The `Group` Attribute of J2EE management MBeans, obtained from `getGroup()`, has a value of `com.sun.appserv.management.base.AMX.GROUP_JSR77`.

Other MBeans

MBeans that do not fit into one of the above four categories have the value `com.sun.appserv.management.base.AMX.GROUP_OTHER`. One such example is `com.sun.appserv.management.deploy.DeploymentMgr`.

MBean Notifications

All AMX MBeans that emit Notifications place a `java.util.Map` within the `userData` field of a standard Notification, which can be obtained via `Notification.getUserData()`. Within the map are zero or more items, which vary according to the Notification type. Each Notification type, and the data available within the Notification, is defined in its respective MBean or in an appropriate place.

Note that certain standard Notifications, such as `javax.management.AttributeChangeNotification` do not and cannot follow this behavior.

Access to MBean Attributes

An AMX MBean `Attribute` is accessible in three ways:

- Dotted names via `MonitoringDottedNames` and `ConfigDottedNames`

- Attributes on MBeans via `getAttribute(s)` and `setAttributes(s)` (from the standard JMX API)
- Getters/setters within the MBean's interface class, for example, `getPort()`, `setPort()`, and so on.

All dotted names that are accessible via the command line interface are available as Attributes within a single MBean. This includes properties, which are Attributes beginning with the prefix `property.`, for example, `server.property.myproperty`.

Note – Certain attributes that may be of a specific type, such as `int`, are declared as `java.lang.String`. This is because the value of the attribute may be a template of a form such as `${HTTP_LISTENER_PORT}`.

Proxies

Proxies are an important part of the AMX API, and enhance ease-of-use for the programmer.

While JMX MBeans can be used directly, client-side proxies are offered to facilitate navigation through the MBean hierarchy. In some cases, proxies also function as support or helper objects to simplify the use of the MBeans.

See the API documentation for the `com.sun.appserv.management` package and its sub-packages for more information about using proxies. The API documentation explains the use of AMX with proxies. If you are using JMX directly (for example, via `MBeanServerConnection`), the return type, argument types and method names might vary as needed for the difference between a strongly-typed proxy interface and generic `MBeanServerConnection/ObjectName` interface.

Connecting to the Domain Administration Server

As stated in "Configuration MBeans" on page 261, the AMX API allows client applications to connect to Application Server instances via the DAS. All AMX connections are established to the DAS only: AMX does not support direct connections to individual server instances. This makes it simple to interact with all servers, clusters, and so on, with a single connection.

Sample code for connecting to the DAS is shown in "Connecting to the DAS" on page 264.

Examining AMX Code Samples

The following example uses of AMX are discussed in this document:

Connecting to the DAS

The connection to the DAS is shown in the following code.

EXAMPLE 16–1 Connecting to the DAS

```
[...]
public static AppserverConnectionSource
    connect(
        final String host,
        final int port,
        final String user,
        final String password,
        final TLSParams tlsParams )
        throws IOException
        {
            final String info = "host=" + host + ", port=" + port +
                ", user=" + user + ", password=" + password +
                ", tls=" + (tlsParams != null);

            SampleUtil.println( "Connecting...:" + info );

            final AppserverConnectionSource conn    =
                new AppserverConnectionSource(
                    AppserverConnectionSource.PROTOCOL_RMI,
                    host, port, user, password, tlsParams, null);

            conn.getJMXConnector( false );

            SampleUtil.println( "Connected: " + info );
```

EXAMPLE 16–1 Connecting to the DAS *(Continued)*

```
            return( conn );
        }
[...]
```

A connection to the DAS is obtained via an instance of the `com.sun.appserv.management.client.AppserverConnectionSource` class. For the connection to be established, you must know the name of the host and port number on which the DAS is running, and have the correct user name, password and TLS parameters.

Once the connection to the DAS is established, `DomainRoot` is obtained as follows:

```
DomainRoot domainRoot = appserverConnectionSource.getDomainRoot();
```

This `DomainRoot` instance is a client-side dynamic proxy to the MBean `amx:j2eeType=X-DomainRoot,name=amx`.

See the API documentation for `com.sun.appserv.management.client.AppserverConnectionSource` for further details about connecting to the DAS using the `AppserverConnectionSource` class.

However, if you prefer to work with standard JMX, instead of getting `DomainRoot`, you can get the `MBeanServerConnection` or `JMXConnector`, as shown:

```
MBeanServerConnection conn =
appserverConnectionSource.getMBeanServerConnection( false );
JMXConnector jmxConn =
appserverConnectionSource.getJMXConnector( false );
```

Starting an Application Server

The `startServer()` method demonstrates how to start an Application Server.

EXAMPLE 16–2 Starting an Application Server

```
[...]
startServer( final String serverName )
{
    final J2EEServer server    = getJ2EEServer( serverName );

    server.start();
}
[...]
```

This method retrieves and starts an application server instance named `server`. The `server` is an instance of the `com.sun.appserv.management.j2see.J2EEServer` interface, and is obtained by calling another method, `getJ2EEServer()`, shown in the following code.

EXAMPLE 16–3 Obtaining a Named J2EE server instance

```
[...]
getJ2EEServer( final String serverName )
{
    final J2EEDomain j2eeDomain = getDomainRoot().getJ2EEDomain();
    final Map servers = j2eeDomain.getServerMap();
    final J2EEServer server = (J2EEServer)servers.get( serverName );
        if ( server == null )
        {
            throw new IllegalArgumentException( serverName );
        }
    return( server );
}
[...]
```

To obtain a J2EE server instance, the `getJ2EEServer()` method first of all obtains an instance of the `J2EEDomain` interface by calling the `com.sun.appserv.management.base.AMX.getDomainRoot()` and `com.sun.appserv.management.DomainRoot.getJ2EEDomain()` methods. The two methods called establish the following:

- `AMX.getDomainRoot()` obtains the Application Server domain to which `j2eeDomain` belongs.
- `DomainRoot.getJ2EEDomain()` obtains the J2EE domain for `j2eeDomain`.

The `J2EEServer` instance is then started by a call to the `start()` method. The `com.sun.appserv.management.j2ee.StateManageable.start()` method can be used to start any state manageable object.

Deploying an Archive

The `uploadArchive()` and `deploy()` methods demonstrate how to upload and deploy a J2EE archive file.

EXAMPLE 16–4 Uploading an archive

```
[...]
uploadArchive ( final File archive  ) throws IOException
{
    final FileInputStream input = new FileInputStream( archive );
```

EXAMPLE 16–4 Uploading an archive *(Continued)*

```
    final long length = input.available();
    final DeploymentMgr mgr = getDomainRoot().getDeploymentMgr();
    final Object uploadID = mgr.initiateFileUpload( length );
    try
    {
        [...]
    }
    finally
    {
        input.close();
    }
    return( uploadID );
}
[...]
```

The `uploadArchive()` method creates a standard Java `FileInputStream` instance called `input`, to upload the archive `archive`. It then obtains the AMX deployment manager running in the application server domain, by calling the `DomainRoot.getDeploymentMgr()` method.

A call to `com.sun.appserv.management.deploy.initiateFileUpload` starts the upload of `archive`. The `initiateFileUpload()` method automatically issues an upload ID, that `uploadArchive()` returns when it is called by `deploy()`.

EXAMPLE 16–5 Deploying an archive

```
[...]
deploy ( final File archive ) throws IOException
{
    final Object uploadID = uploadArchive(archive);
    final DeploymentMgr mgr    = getDomainRoot().getDeploymentMgr();
    final Object deployID = mgr.initDeploy( );
    final DeployNotificationListener myListener =
        new DeployNotificationListener( deployID);
    mgr.addNotificationListener( myListener, null, null);
    try
    {
        final Map    options  = new HashMap();
        options.put( DeploymentMgr.DEPLOY_OPTION_VERIFY_KEY,
            Boolean.TRUE.toString() );
        options.put( DeploymentMgr.DEPLOY_OPTION_DESCRIPTION_KEY,
            "description" );
        mgr.startDeploy( deployID, uploadID, null, null);
        while ( ! myListener.isCompleted() )
        {
            try
```

EXAMPLE 16–5 Deploying an archive *(Continued)*

```
            {
                println( "deploy: waiting for deploy of " + archive);
                Thread.sleep( 1000 );
            }
            catch( InterruptedException e )
            {
            }
        }
        final DeploymentStatus status = myListener.getDeploymentStatus();
        println( "Deployment result: " + getStageStatusString(
            status.getStageStatus() ) );
        if ( status.getStageThrowable() != null )
        {
            status.getStageThrowable().printStackTrace();
        }
    }
    finally
    {
        try
        {
            mgr.removeNotificationListener( myListener );
        }
        catch( Exception e )
        {
        }
    }
}
[...]
```

The `deploy()` method calls `uploadArchive` to get the upload ID for `archive`. It then identifies the deployment manager by calling `DomainRoot.getDeploymentMgr()`. A call to `DeploymentMgr.initDeploy()` initializes the deployment and obtains a deployment ID, which is used to track the progress of the deployment.

A JMX notification listener, `myListener`, is created and activated to listen for notifications regarding the deployment of `deployID`.

Deployment is started by calling the `DeploymentMgr.startDeploy()` method and providing it with the `deployID` and `uploadID`.

While the deployment is continuing, `myListener` listens for the completion notification and `DeploymentStatus` keeps you informed of the status of the deployment by regularly calling its `getStageStatus()` method. Once the deployment is complete, the listener is closed down.

Caution – Some of the behavior of the `com.sun.appserv.management.deploy` API is unpredictable, and it should be used with caution.

Displaying the AMX MBean Hierarchy

The `displayAMX()` method demonstrates how to display the AMX MBean hierarchy.

EXAMPLE 16–6 Displaying the AMX MBean Hierarchy

```
[...]
displayAMX(
    final AMX amx,
    final int indentCount )
{
    final String indent = getIndent( indentCount );
    final String j2eeType = amx.getJ2EEType();
    final String name = amx.getName();
    if ( name.equals( AMX.NO_NAME ) )
    {
        println( indent + j2eeType );
    }
    else
    {
        println( indent + j2eeType + "=" + name );
    }
}
private void
displayHierarchy(
    final Collection amxSet,
    final int indentCount )
{
    final Iterator    iter    = amxSet.iterator();
    while ( iter.hasNext() )
    {
        final AMX amx = (AMX)iter.next();
        displayHierarchy( amx, indentCount );
    }
}
public void
displayHierarchy(
    final AMX amx,
    final int    indentCount )
{
    displayAMX( amx, indentCount );
    if ( amx instanceof Container )
```

EXAMPLE 16–6 Displaying the AMX MBean Hierarchy *(Continued)*

```
    {
        final Map m = ((Container)amx).getMultiContaineeMap( null );
        final Set deferred = new HashSet();
        final Iterator mapsIter = m.values().iterator();
        while ( mapsIter.hasNext() )
        {
            final Map instancesMap = (Map)mapsIter.next();
            final AMX first = (AMX)instancesMap.values().iterator().next();
            if ( first instanceof Container )
            {
                deferred.add( instancesMap );
            }
            else
            {
                displayHierarchy( instancesMap.values(), indentCount + 2);
            }
        }
        // display deferred items
        final Iterator iter = deferred.iterator();
        while ( iter.hasNext() )
        {
            final Map instancesMap = (Map)iter.next();
            displayHierarchy( instancesMap.values(), indentCount + 2);
        }
    }
}
public void displayHierarchy()
{
    displayHierarchy( getDomainRoot(), 0);
}
public void
displayHierarchy( final String j2eeType )
{
    final Set items = getQueryMgr().queryJ2EETypeSet( j2eeType );
    if ( items.size() == 0 )
    {
        println( "No {@link AMX} of j2eeType "
            + SampleUtil.quote( j2eeType ) + " found" );
    }
    else
    {
        displayHierarchy( items, 0);
    }
}
[...]
```

The `displayAMX()` method obtains the J2EE type and the name of an AMX MBean by calling `AMX.getJ2EEType` and `AMX.getName` respectively.

The `displayHierarchy()` method defines a standard Java `Collection` instance, `amxSet`, which collects instances of AMX MBeans.

To display the hierarchy of MBeans within a particular MBean in the collection, `displayHierarchy()` checks whether the MBean is an instance of `Container`. If so, it creates a set of the MBeans it contains by calling the `com.sun.appserv.management.base.Container.getMultiContaineeMap()` method.

The MBean hierarchy for a particular J2EE type is displayed by calling the `com.sun.appserv.management.base.QueryMgr.queryJ2EETypeSet()`, and passing the result to `displayHierarchy()`.

To display the entire AMX MBean hierarchy in a domain, `displayHierarchy()` calls `getDomainRoot()` to obtain the root AMX MBean in the domain.

Setting Monitoring States

The `setMonitoring()` method demonstrates how to set monitoring states.

EXAMPLE 16–7 Setting Monitoring States

```
[...]
private static final Set LEGAL_MON =
    Collections.unmodifiableSet( SampleUtil.newSet( new String[]
{
    ModuleMonitoringLevelValues.HIGH,
    ModuleMonitoringLevelValues.LOW,
    ModuleMonitoringLevelValues.OFF,
} ));
public void setMonitoring(
    final String configName,
    final String state )
{
    if ( ! LEGAL_MON.contains( state ) )
    {
        throw new IllegalArgumentException( state );
    }
    final ConfigConfig config =
        (ConfigConfig)getDomainConfig().
        getConfigConfigMap().get( configName );
    final ModuleMonitoringLevelsConfig mon =
        config.getMonitoringServiceConfig().
        getModuleMonitoringLevelsConfig();
```

EXAMPLE 16–7 Setting Monitoring States *(Continued)*

```
    mon.setConnectorConnectionPool( state );
    mon.setThreadPool( state );
    mon.setHTTPService( state );
    mon.setJDBCConnectionPool( state );
    mon.setORB( state );
    mon.setTransactionService( state );
    mon.setWebContainer( state );
    mon.setEJBContainer( state );
}
[...]
```

The AMX API defines three levels of monitoring in `com.sun.appserv.management.config.ModuleMonitoringLevelValues`, namely, `HIGH`, `LOW`, and `OFF`.

In this example, the configuration element being monitored is named `configName`. The `com.sun.appserv.management.config.ConfigConfig` interface is used to configure the `config` element for `configName` in the `domain.xml` file.

An instance of `com.sun.appserv.management.config.ModuleMonitoringLevelsConfig` is created to configure the `module-monitoring-levels` element for `configName` in the `domain.xml` file.

The `ModuleMonitoringLevelsConfig` instance created then calls each of its set methods to change their states to `state`.

The above is performed by running the `set-monitoring` command when you run `SimpleMain`, stating the name of the configuration element to be monitored and the monitoring `state` to one of `HIGH`, `LOW` or `OFF`.

Accessing AMX MBeans

The `handleList()` method demonstrates how to access many (but not all) configuration elements.

EXAMPLE 16–8 Accessing AMX MBeans

```
[...]
handleList()
{
    final DomainConfig dcp = getDomainConfig();
    println( "\n--- Top-level --- \n" );
    displayMap( "ConfigConfig", dcp.getConfigConfigMap() );
```

EXAMPLE 16–8 Accessing AMX MBeans *(Continued)*

```
    displayMap( "ServerConfig", dcp.getServerConfigMap() );
    displayMap( "StandaloneServerConfig",
        dcp.getStandaloneServerConfigMap() );
    displayMap( "ClusteredServerConfig",
        dcp.getClusteredServerConfigMap() );
    displayMap( "ClusterConfig", dcp.getClusterConfigMap() );
    println( "\n--- DeployedItems --- \n" );
    displayMap( "J2EEApplicationConfig",
        dcp.getJ2EEApplicationConfigMap() );
    displayMap( "EJBModuleConfig",
        dcp.getEJBModuleConfigMap() );
    displayMap( "WebModuleConfig",
        dcp.getWebModuleConfigMap() );
    displayMap( "RARModuleConfig",
        dcp.getRARModuleConfigMap() );
    displayMap( "AppClientModuleConfig",
        dcp.getAppClientModuleConfigMap() );
    displayMap( "LifecycleModuleConfig",
        dcp.getLifecycleModuleConfigMap() );
    println( "\n--- Resources --- \n" );
    displayMap( "CustomResourceConfig",
        dcp.getCustomResourceConfigMap() );
    displayMap( "PersistenceManagerFactoryResourceConfig",
        dcp.getPersistenceManagerFactoryResourceConfigMap() );
    displayMap( "JNDIResourceConfig",
        dcp.getJNDIResourceConfigMap() );
    displayMap( "JMSResourceConfig",
        dcp.getJMSResourceConfigMap() );
    displayMap( "JDBCResourceConfig",
        dcp.getJDBCResourceConfigMap() );
    displayMap( "ConnectorResourceConfig",
        dcp.getConnectorResourceConfigMap() );
    displayMap( "JDBCConnectionPoolConfig",
        dcp.getJDBCConnectionPoolConfigMap() );
    displayMap( "PersistenceManagerFactoryResourceConfig",
        dcp.getPersistenceManagerFactoryResourceConfigMap() );
    displayMap( "ConnectorConnectionPoolConfig",
        dcp.getConnectorConnectionPoolConfigMap() );
    displayMap( "AdminObjectResourceConfig",
        dcp.getAdminObjectResourceConfigMap() );
    displayMap( "ResourceAdapterConfig",
        dcp.getResourceAdapterConfigMap() );
    displayMap( "MailResourceConfig",
        dcp.getMailResourceConfigMap() );
    final ConfigConfig config =
        (ConfigConfig)dcp.getConfigConfigMap().get( "server-config" );
```

EXAMPLE 16–8 Accessing AMX MBeans *(Continued)*

```
    println( "\n--- HTTPService --- \n" );
    final HTTPServiceConfig httpService = config.getHTTPServiceConfig();
    displayMap( "HTTPListeners",
        httpService.getHTTPListenerConfigMap() );
    displayMap( "VirtualServers",
        httpService.getVirtualServerConfigMap() );
}
[...]
```

The `handleList()` method makes use of the `displayMap()` method, which simply prints out the key value pairs.

The `handleList()` method identifies the configuration for a domain by calling the `DomainRoot.getDomainConfig()` method. This `DomainConfig` instance then calls each of its `getXXXMap()` methods in turn, to obtain a `Map` for each type of AMX MBean. The `Map` returned by each getter is displayed by `displayMap()`.

Similarly, the AMX MBeans representing the `http-service` element are displayed as `Maps` by calling the `getXXXMap()` methods of the `com.sun.appserv.management.config.HTTPServiceConfig` interface, and passing them to `displayMap()`.

Accessing and Displaying the Attributes of an AMX MBean

The `displayAllAttributes()` method demonstrates how to access and display the attributes of an AMX MBean.

EXAMPLE 16–9 Accessing and Displaying the Attributes of an AMX MBean

```
[...]
displayAllAttributes( final AMX item )
{
    println( "\n--- Attributes for " + item.getJ2EEType() +
        "=" + item.getName() + " ---" );
    final Extra extra = Util.getExtra( item );
    final Map attrs    = extra.getAllAttributes();
    final Iterator iter = attrs.keySet().iterator();
    while ( iter.hasNext() )
    {
        final String name = (String)iter.next();
        final Object value = attrs.get( name );
        println( name + "=" + toString( value ) );
```

EXAMPLE 16–9 Accessing and Displaying the Attributes of an AMX MBean *(Continued)*

```
    }
}
public void
displayAllAttributes( final String j2eeType )
{
    final Set items = queryForJ2EEType( j2eeType );
    if ( items.size() == 0 )
    {
        println( "No {@link AMX} of j2eeType "
            + SampleUtil.quote( j2eeType ) + " found" );
    }
    else
    {
        final Iterator iter= items.iterator();
        while ( iter.hasNext() )
        {
            final AMX amx = (AMX)iter.next();
            displayAllAttributes( amx );
            println( "" );
        }
    }
}
[...]
```

The `displayAllAttributes()` method calls the `AMX.getName()` and `AMX.getJ2EEType()` methods for an AMX MBean and prints the results onscreen. It then gets all the attributes for that MBean by calling `com.sun.appserv.management.base.Extra.getAllAttributes()` on the `Extra` instance returned by `com.sun.appserv.management.base.Util.getExtra()`. This is repeated for every MBean.

The attributes of AMX MBeans of a certain J2EE type can be displayed by specifying the J2EE type when the command is run. In this case, `displayAllAttributes()` calls `queryForJ2EEType()`. The `queryForJ2EEType()` method calls the `com.sun.appserv.management.base.QueryManager.queryPropSet()` method on the specified J2EE type to identify all elements of that type in the domain.

Listing AMX MBean Properties

The `displayAllProperties()` demonstrates how to list AMX MBean properties.

EXAMPLE 16–10 Listing AMX MBean Properties

```
[...]
getProperties( final PropertiesAccess pa )
{
    final HashMap m = new HashMap();
    final String[] names = pa.getPropertyNames();
    for( int i = 0; i < names.length; ++i )
    {
        m.put( names[ i ], pa.getPropertyValue( names[ i ] ) );
    }
    return( m );
}
public void
displayAllProperties( )
{
    final Iterator iter    = getQueryMgr().queryAllSet().iterator();
    while ( iter.hasNext() )
    {
        final AMX amx = (AMX)iter.next();
        if ( amx instanceof PropertiesAccess )
        {
            final PropertiesAccess pa = (PropertiesAccess)amx;
            final Map    props    = getProperties( pa );
            if ( props.keySet().size() != 0 )
            {
                println( "\nProperties for:
                    " + Util.getObjectName( AMX)pa ) );
                println( SampleUtil.mapToString(getProperties(pa), "\n") );
            }
        }
    }
}
[...]
```

The `displayAllProperties()` method uses another `Samples` method, `getProperties()`. This method creates an instance of the `com.sun.appserv.management.config.PropertiesAccess` interface, and calls its `getPropertyNames()` method to obtain the names of all the properties for a given AMX MBean. For each property name obtained, its corresponding value is obtained by calling `PropertiesAccess.getPropertyValue()`.

The `displayAllProperties()` method calls the `com.sun.appserv.management.base.QueryMgr.queryAllSet()` method to obtain a set of all the AMX MBeans present in the domain. All AMX MBeans that have properties obligatorily extend the `PropertiesAccess` interface. Any MBean found to extend `PropertiesAccess` is passed to the `getProperties()` method, and the list of property values returned is printed onscreen.

Querying

The `demoQuery()` method demonstrates how to issue queries.

The `demoQuery()` method uses other methods that are defined by `Samples`, namely `displayWild()`, and `displayJ2EEType()`. The `displayWild()` method is shown in the following code.

EXAMPLE 16–11 Querying and displaying wild cards

```
[...]
queryWild(
    final String propertyName,
    final String propertyValue)
{
    final String[] propNames = new String[] { propertyName };
    final String[] propValues = new String[]{ propertyValue };
    final Set amxs = getQueryMgr().queryWildSet( propNames, propValues );
    return( amxs );
}
public Set
displayWild(
    final String propertyName,
    final String propertyValue)
{
    final Set items = queryWild( propertyName, propertyValue );
    println( "\n--- Queried for " + propertyName + "="
        + propertyValue + " ---" );
    final Iterator    iter    = items.iterator();
    while ( iter.hasNext() )
    {
        final AMX    item    = (AMX)iter.next();
        println( "j2eeType=" + item.getJ2EEType() + ",
            " + "name=" + item.getName() );
    }
}
[...]
```

The `displayWild()` method calls `queryWild()`, to obtain all the AMX MBeans that have object names matching `propertyName` and `propertyValue`. To do so, `queryWild()` calls the `com.sun.appserv.management.base.QueryMgr.queryWildSet()` method. The `queryWildSet()` method returns the list of AMX MBeans with object names matching the wild card strings.

For each MBean returned, the `displayWild()` calls `AMX.getJ2EEType()` to identify its J2EE type, and prints the result onscreen.

In code that is not shown here, the `displayJ2EEType()` method calls the `queryForJ2EEType()` method, which was seen in "Accessing and Displaying the Attributes of an AMX MBean" on page 274, to identify MBeans of a certain J2EE type and print their object names onscreen.

EXAMPLE 16–12 Querying

```
[...]
demoQuery()
{
    displayWild( AMX.J2EE_TYPE_KEY, "X-*ResourceConfig" );
    displayWild( AMX.J2EE_TYPE_KEY, "X-*ServerConfig" );
    displayJ2EEType( XTypes.SSL_CONFIG );
    displayJ2EEType( XTypes.CLUSTER_CONFIG );
}
[...]
```

In the `demoQuery()` method, the `displayWild()` and `displayJ2EEType()` methods are called to find the following MBeans:

- `J2EE_TYPE_KEY` MBeans called `ResourceConfig`
- `J2EE_TYPE_KEY` MBeans called `ServerConfig`
- All `SSL_CONFIG` MBeans
- All `CLUSTER_CONFIG` MBeans

Monitoring Attribute Changes

The `demoJMXMonitor()` demonstrates how to monitor attribute changes.

EXAMPLE 16–13 Monitoring Attribute Changes

```
[...]
demoJMXMonitor() throws InstanceNotFoundException, IOException
{
    final JMXMonitorMgr mgr = getDomainRoot().getJMXMonitorMgr();
    final String attrName = "SampleString";
    final String attrValue = "hello";
    final SampleListener sampleListener = new SampleListener();
    final MBeanServerConnection conn =
        Util.getExtra( mgr ).getConnectionSource()
        .getExistingMBeanServerConnection();
    conn.addNotificationListener(
        getMBeanServerDelegateObjectName(),
        sampleListener, null, null );
    final Sample sample = (Sample)getDomainRoot()
        .getContainee( XTypes.SAMPLE );
    final String monitorName = "SampleStringMonitor";
```

EXAMPLE 16–13 Monitoring Attribute Changes *(Continued)*

```
    AMXStringMonitor mon = null;
    try
    {
        try { mgr.remove( monitorName ); }
        catch( Exception e ) {}
        mon = mgr.createStringMonitor( monitorName );
        waitMBeanServerNotification( sampleListener,
            MBeanServerNotification.REGISTRATION_NOTIFICATION,
            Util.getObjectName( mon ) );
        sample.addAttribute( attrName, attrValue );
        mon.addNotificationListener( sampleListener, null, null);
        mon.setObservedAttribute( attrName );
        mon.setStringToCompare( attrValue );
        mon.setNotifyDiffer( true );
        mon.setNotifyMatch( true );
        mon.addObservedObject( Util.getObjectName( sample ) );
        final StdAttributesAccess attrs = Util.getExtra( sample);
        attrs.setAttribute( new Attribute(attrName, "goodbye") );
        attrs.setAttribute( new Attribute(attrName, attrValue) );
        sample.removeAttribute( attrName );
        final Map notifs = sampleListener.getNotifsReceived();
        waitNumNotifs( notifs,
            AttributeChangeNotification.ATTRIBUTE_CHANGE, 4 );
    }
    catch( Throwable t )
    {
        t.printStackTrace();
    }
    finally
    {
        try
        {
            mon.removeNotificationListener( sampleListener );
            if ( mon != null )
            {
                mgr.remove( mon.getName() );
                waitMBeanServerNotification( sampleListener,
                    MBeanServerNotification
                    .UNREGISTRATION_NOTIFICATION,
                    Util.getObjectName( mon ) );
            }
            conn.removeNotificationListener(
            getMBeanServerDelegateObjectName(),
            sampleListener );
        }
        catch( ListenerNotFoundException e )
```

EXAMPLE 16–13 Monitoring Attribute Changes *(Continued)*

```
        {
        }
    }
}
[...]
```

The `demoJmx()` method demonstrates the implementation of a JMX monitor MBean, that listens for changes in a certain attribute. This is achieved in the following stages:

1. A `com.sun.appserv.management.monitor.JMXMonitorMgr` instance is obtained using the `DomainRoot.getJMXMonitorMgr()` method.
2. A `SampleListener` JMX notification listener that is provided in the sample package is instantiated.
3. A connection to the domain's MBean server is obtained by calling `com.sun.appserv.management.client.ConnectionSource.getExistingMBeanServerConnection()` on the `JMXMonitorMgr` instance's `Extra` information.
4. The `SampleListener` notification listener is added to the MBean server connection, with an MBean server delegate obtained from `getMBeanServerDelegateObject()`. The notification listener is now in place on the MBean server connection.
5. An AMX MBean, `sample`, of the type `SAMPLE` is obtained by calling the `com.sun.appserv.management.base.Container.getContainee()` method on an instance of the `Sample` interface. The `Sample` interface defines a basic AMX MBean.
6. An `AMXStringMonitor`, an AMX-compatible JMX `StringMonitorMBean`, is instantiated by calling `createStringMonitor` on the `JMXMonitorMgr` instance created above. The `AMXStringMonitor` instance then calls `waitMBeanServerNotification()`. The `waitMBeanServerNotification()` method waits for MBean server notifications of the type `REGISTRATION_NOTIFICATION` from the `SampleListener` instance that is listening on the MBean server connection.
7. An attribute of name `attrName` and value `attrValue` is added to the AMX MBean `sample`.
8. Various methods of the `AMXStringMonitor` instance are called, to add a listener, and to set the value to be observed, the object to be observed, and so on.
9. Access to the `sample` MBean's attributes is obtained by passing the `sample` MBean's `Extra` information to an instance of `com.sun.appserv.management.base.StdAttributesAccess`. The `StdAttributesAccess.setAttribute()` method is then called to change the values of these attributes.

10. The `AMXStringMonitor` then calls the sample notification listener's `getNotifsReceived()` method to retrieve the notifications that resulted from the calls to `setAttribute()` above. The `waitNumNotifs()` method waits until four `ATTRIBUTE_CHANGE` notifications have been received before exiting.
11. The notification listener is then removed and the monitor is closed down.

Undeploying Modules

The `undeploy()` method demonstrates how to undeploy a module.

EXAMPLE 16–14 Undeploying Modules

```
[...]
undeploy ( final String moduleName ) throws IOException
{
    final DeploymentMgr mgr = getDomainRoot().getDeploymentMgr();

    final Map statusData = mgr.undeploy( moduleName, null );
    final DeploymentStatus status =
        DeploymentSupport.mapToDeploymentStatus( statusData );
    println( "Undeployment result: "
        + getStageStatusString(status.getStageStatus()));
    if ( status.getStageThrowable() != null )
    {
        status.getStageThrowable().printStackTrace();
    }
}
[...]
```

The `undeploy()` method obtains the `DeploymentMgr` instance for the domain in the same way that `deploy()` does so. It then calls the `DeploymentMgr.undeploy()` method for a named module.

Stopping an Application Server

The `stopServer()` method demonstrates how to stop an application server. The `stopServer()` method simply calls the `getJ2EEServer()` method on a given server instance, and then calls `J2EEServer.stop()`.

Running the AMX Samples

To set up your development environment for using AMX, you must ensure that your Java classpath contains the following Java archive (JAR) files:

- `appserv-admin.jar` - The JAR file containing the AMX interfaces needed for your client. This file is found in *install-dir*`/lib/`. No other classes from this JAR file should be used by your program.
- `jmxri.jar` - The runtime libraries for the JMX Reference Implementation. If you are using JDK 1.5, these are already in the JDK.
- `jmxremote.jar` - The runtime libraries for the JMX Remote API. If you are using JDK 1.5, these are already in the JDK.
- `j2ee.jar` - The runtime libraries for the J2EE Platform. This file is found in *install-dir*`/lib/`. This JAR file is needed only if you intend to use any of the J2EE Management Statistic classes (`javax.management.j2ee.*`).

Start your Java application in a manner similar to this:

```
export JAR_PATH=install-dir/lib/
export CP="$JAR_PATH/j2ee.jar:$JAR_PATH/appserv-admin.jar"
java -cp $CP com.mycompany.MyClientMain
```

◆ ◆ ◆ APPENDIX A

Deployment Descriptor Files

This chapter describes deployment descriptor files specific to the Sun Java System Application Server in the following sections:

Sun Java System Application Server Descriptors

Sun Java System Application Server uses deployment descriptors in addition to the J2EE standard descriptors for configuring features specific to the Application Server. The `sun-application.xml`, `sun-web.xml`, and `sun-cmp-mappings.xml` files are optional; all the others are required.

Note – Settings in the Application Server deployment descriptors override corresponding settings in the Java EE deployment descriptors and in the Application Server's `domain.xml` file unless otherwise stated. For more information about the `domain.xml` file, see the *Sun Java System Application Server Platform Edition 8.2 Administration Reference*.

Each deployment descriptor (or XML) file has a corresponding DTD file, which defines the elements, data, and attributes that the deployment descriptor file can contain. For example, the `sun-application_1_4-0.dtd` file defines the structure of the `sun-application.xml` file. The DTD files for the Application Server deployment descriptors are located in the *install-dir*`/lib/dtds` directory.

Note – Do not edit the DTD files; their contents change only with new versions of the Application Server.

To check the correctness of these deployment descriptors prior to deployment, see "The Deployment Descriptor Verifier" on page 84.

For general information about DTD files and XML, see the XML specification at `http://www.w3.org/TR/REC-xml`.

The following table lists the Application Server deployment descriptors and their DTD files.

TABLE A–1 Sun Java System Application Server Descriptors

Deployment Descriptor	DTD File	Description
`sun-application.xml`	`sun-application_1_4-0.dtd`	Configures an entire J2EE application (EAR file).
`sun-web.xml`	`sun-web-app_2_4-1.dtd`	Configures a web application (WAR file).
`sun-ejb-jar.xml`	`sun-ejb-jar_2_1-1.dtd`	Configures an enterprise bean (EJB JAR file).
`sun-cmp-mappings.xml`	`sun-cmp-mapping_1_2.dtd`	Configures container-managed persistence for an enterprise bean.
`sun-application-client.xml`	`sun-application-client_1_4-1.dtd`	Configures an Application Client Container (ACC) client (JAR file).
`sun-acc.xml`	`sun-application-client-container_1_0.dtd`	Configures the Application Client Container.

Note – The Application Server deployment descriptors must be readable and writable by the file owners.

In each deployment descriptor file, subelements must be defined in the order in which they are listed under each **Subelements** heading, unless otherwise noted.

The sun-application.xml File

The element hierarchy in the `sun-application.xml` file is as follows:

```
sun-application
.  web
.  .  web-uri
.  .  context-root
.  pass-by-reference
.  unique-id
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  realm
```

Here is a sample `sun-application.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-application PUBLIC '-//Sun Microsystems, Inc.//DTD Application
Server 8.1 J2EE Application 1.4//EN'
'http://www.sun.com/software/appserver/dtds/sun-application_1_4-0.dtd'>
<sun-application>
   <unique-id>67488732739338240</unique-id>
</sun-application>
```

The sun-web.xml File

The element hierarchy in the `sun-web.xml` file is as follows:

```
sun-web-app
.  context-root
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  servlet
.  .  servlet-name
.  .  principal-name
.  .  webservice-endpoint
.  .  .  port-component-name
.  .  .  endpoint-address-uri
.  .  .  login-config
.  .  .  .  auth-method
.  .  .  message-security-binding
.  .  .  .  message-security
```

```
. . . . . message
. . . . . . java-method
. . . . . . . method-name
. . . . . . . method-params
. . . . . . . . method-param
. . . . . . operation-name
. . . . . request-protection
. . . . . response-protection
. . . transport-guarantee
. . . service-qname
. . . tie-class
. . . servlet-impl-class
. idempotent-url-pattern
. session-config
. . session-manager
. . . manager-properties
. . . . property (with attributes)
. . . . . description
. . . store-properties
. . . . property (with attributes)
. . . . . description
. . session-properties
. . . property (with attributes)
. . . . description
. . cookie-properties
. . . property (with attributes)
. . . . description
. ejb-ref
. . ejb-ref-name
. . jndi-name
. resource-ref
. . res-ref-name
. . jndi-name
. . default-resource-principal
. . . name
. . . password
. resource-env-ref
. . resource-env-ref-name
. . jndi-name
. service-ref
. . service-ref-name
. . port-info
. . . service-endpoint-interface
. . . wsdl-port
. . . . namespaceURI
. . . . localpart
. . . stub-property
. . . . name
```

```
. . . . value
. . . call-property
. . . . name
. . . . value
. . . message-security-binding
. . . . message-security
. . . . . message
. . . . . . java-method
. . . . . . . method-name
. . . . . . . method-params
. . . . . . . . method-param
. . . . . . operation-name
. . . . . request-protection
. . . . . response-protection
. . call-property
. . . name
. . . value
. . wsdl-override
. . service-impl-class
. . service-qname
. . . namespaceURI
. . . localpart
. cache
. . cache-helper
. . . property (with attributes)
. . . . description
. . default-helper
. . . property (with attributes)
. . . . description
. . property (with attributes)
. . . description
. . cache-mapping
. . . servlet-name
. . . url-pattern
. . . cache-helper-ref
. . . dispatcher
. . . timeout
. . . refresh-field
. . . http-method
. . . key-field
. . . constraint-field
. . . . constraint-field-value
. class-loader
. . property (with attributes)
. . . description
. jsp-config
. locale-charset-info
. . locale-charset-map
```

```
.  .  parameter-encoding
.  property (with attributes)
.  .  description
.  parameter-encoding
.  message-destination
.  .  message-destination-name
.  .  jndi-name
.  webservice-description
.  .  webservice-description-name
.  .  wsdl-publish-location
```

Here is a sample `sun-web.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Application
Server 8.1 Servlet 2.4//EN'
'http://www.sun.com/software/appserver/dtds/sun-web-app_2_4-1.dtd'>
<sun-web-app>
   <session-config>
      <session-manager/>
   </session-config>
   <resource-ref>
      <res-ref-name>mail/Session</res-ref-name>
      <jndi-name>mail/Session</jndi-name>
   </resource-ref>
   <jsp-config/>
</sun-web-app>
```

The sun-ejb-jar.xml File

The element hierarchy in the `sun-ejb-jar.xml` file is as follows:

```
sun-ejb-jar
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  enterprise-beans
.  .  name
.  .  unique-id
.  .  ejb
.  .  .  ejb-name
.  .  .  jndi-name
.  .  .  ejb-ref
.  .  .  .  ejb-ref-name
.  .  .  .  jndi-name
```

```
. . . resource-ref
. . . . res-ref-name
. . . . jndi-name
. . . . default-resource-principal
. . . . . name
. . . . . password
. . . resource-env-ref
. . . . resource-env-ref-name
. . . . jndi-name
. . . service-ref
. . . . service-ref-name
. . . . port-info
. . . . . service-endpoint-interface
. . . . . wsdl-port
. . . . . . namespaceURI
. . . . . . localpart
. . . . . stub-property
. . . . . . name
. . . . . . value
. . . . . call-property
. . . . . . name
. . . . . . value
. . . . . message-security-binding
. . . . . . message-security
. . . . . . . message
. . . . . . . . java-method
. . . . . . . . . method-name
. . . . . . . . . method-params
. . . . . . . . . . method-param
. . . . . . . . operation-name
. . . . . . . request-protection
. . . . . . . response-protection
. . . . call-property
. . . . . name
. . . . . value
. . . . wsdl-override
. . . . service-impl-class
. . . . service-qname
. . . . . namespaceURI
. . . . . localpart
. . . pass-by-reference
. . . cmp
. . . . mapping-properties
. . . . is-one-one-cmp
. . . . one-one-finders
. . . . . finder
. . . . . . method-name
. . . . . . query-params
```

```
. . . . . . query-filter
. . . . . . query-variables
. . . . . . query-ordering
. . . . prefetch-disabled
. . . . . query-method
. . . . . . method-name
. . . . . . method-params
. . . . . . . method-param
. . . principal
. . . . name
. . . mdb-connection-factory
. . . . jndi-name
. . . . default-resource-principal
. . . . . name
. . . . . password
. . . jms-durable-subscription-name
. . . jms-max-messages-load
. . . ior-security-config
. . . . transport-config
. . . . . integrity
. . . . . confidentiality
. . . . . establish-trust-in-target
. . . . . establish-trust-in-client
. . . . as-context
. . . . . auth-method
. . . . . realm
. . . . . required
. . . . sas-context
. . . . . caller-propagation
. . . is-read-only-bean
. . . refresh-period-in-seconds
. . . commit-option
. . . cmt-timeout-in-seconds
. . . use-thread-pool-id
. . . gen-classes
. . . . remote-impl
. . . . local-impl
. . . . remote-home-impl
. . . . local-home-impl
. . . bean-pool
. . . . steady-pool-size
. . . . resize-quantity
. . . . max-pool-size
. . . . pool-idle-timeout-in-seconds
. . . . max-wait-time-in-millis
. . . bean-cache
. . . . max-cache-size
. . . . resize-quantity
```

```
. . . . is-cache-overflow-allowed
. . . . cache-idle-timeout-in-seconds
. . . . removal-timeout-in-seconds
. . . . victim-selection-policy
. . . mdb-resource-adapter
. . . . resource-adapter-mid
. . . . activation-config
. . . . . description
. . . . . activation-config-property
. . . . . . activation-config-property-name
. . . . . . activation-config-property-value
. . . webservice-endpoint
. . . . port-component-name
. . . . endpoint-address-uri
. . . . login-config
. . . . . auth-method
. . . . message-security-binding
. . . . . message-security
. . . . . . message
. . . . . . . java-method
. . . . . . . . method-name
. . . . . . . . method-params
. . . . . . . . . method-param
. . . . . . . operation-name
. . . . . . request-protection
. . . . . . response-protection
. . . . transport-guarantee
. . . . service-qname
. . . . tie-class
. . . . servlet-impl-class
. . . flush-at-end-of-method
. . . . method
. . . . . description
. . . . . ejb-name
. . . . . method-name
. . . . . method-intf
. . . . . method-params
. . . . . . method-param
. . . checkpointed-methods
. . . checkpoint-at-end-of-method
. . . . method
. . . . . description
. . . . . ejb-name
. . . . . method-name
. . . . . method-intf
. . . . . method-params
. . . . . . method-param
. . pm-descriptors
```

```
. . cmp-resource
. . . jndi-name
. . . default-resource-principal
. . . . name
. . . . password
. . . property (with subelements)
. . . . name
. . . . value
. . . create-tables-at-deploy
. . . drop-tables-at-undeploy
. . . database-vendor-name
. . . schema-generator-properties
. . . . property (with subelements)
. . . . . name
. . . . . value
. . message-destination
. . . message-destination-name
. . . jndi-name
. . webservice-description
. . . webservice-description-name
. . . wsdl-publish-location
```

Note – If any configuration information for an enterprise bean is not specified in the `sun-ejb-jar.xml` file, it defaults to a corresponding setting in the EJB container if an equivalency exists.

Here is a sample `sun-ejb-jar.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Application
Server 8.1 EJB 2.1//EN'
'http://www.sun.com/software/appserver/dtds/sun-ejb-jar_2_1-1.dtd'>
<sun-ejb-jar>
<display-name>First Module</display-name>
<enterprise-beans>
    <ejb>
        <ejb-name>CustomerEJB</ejb-name>
        <jndi-name>customer</jndi-name>
        <bean-pool>
            <steady-pool-size>10</steady-pool-size>
            <resize-quantity>10</resize-quantity>
            <max-pool-size>100</max-pool-size>
            <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
        </bean-pool>
        <bean-cache>
            <max-cache-size>100</max-cache-size>
```

```
            <resize-quantity>10</resize-quantity>
            <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
            <victim-selection-policy>LRU</victim-selection-policy>
        </bean-cache>
    </ejb>
    <cmp-resource>
        <jndi-name>jdbc/__default</jndi-name>
        <create-tables-at-deploy>true</create-tables-at-deploy>
        <drop-tables-at-undeploy>true</drop-tables-at-undeploy>
    </cmp-resource>
</enterprise-beans>
</sun-ejb-jar>
```

The sun-cmp-mappings.xml File

The element hierarchy in the `sun-cmp-mappings.xml` file is as follows:

```
sun-cmp-mappings
.  sun-cmp-mapping
.  .  schema
.  .  entity-mapping
.  .  .  ejb-name
.  .  .  table-name
.  .  .  cmp-field-mapping
.  .  .  .  field-name
.  .  .  .  column-name
.  .  .  .  read-only
.  .  .  .  fetched-with
.  .  .  .  .  default
.  .  .  .  .  level
.  .  .  .  .  named-group
.  .  .  .  .  none
.  .  .  cmr-field-mapping
.  .  .  .  cmr-field-name
.  .  .  .  column-pair
.  .  .  .  .  column-name
.  .  .  .  fetched-with
.  .  .  .  .  default
.  .  .  .  .  level
.  .  .  .  .  named-group
.  .  .  .  .  none
.  .  .  secondary-table
.  .  .  .  table-name
.  .  .  .  column-pair
.  .  .  .  .  column-name
.  .  .  consistency
```

```
. . . . none
. . . . check-modified-at-commit
. . . . lock-when-loaded
. . . . check-all-at-commit
. . . . lock-when-modified
. . . . check-version-of-accessed-instances
. . . . . column-name
```

Here is a sample database schema definition:

```
create table TEAMEJB (
TEAMID varchar2(256) not null,
 NAME varchar2(120) null,
 CITY char(30) not null,
 LEAGUEEJB_LEAGUEID varchar2(256) null,
 constraint PK_TEAMEJB primary key (TEAMID)
)
create table PLAYEREJB (
POSITION varchar2(15) null,
 PLAYERID varchar2(256) not null,
 NAME char(64) null,
 SALARY number(10, 2) not null,
 constraint PK_PLAYEREJB primary key (PLAYERID)
)
create table LEAGUEEJB (
LEAGUEID varchar2(256) not null,
 NAME varchar2(256) null,
 SPORT varchar2(256) null,
 constraint PK_LEAGUEEJB primary key (LEAGUEID)
)
create table PLAYEREJBTEAMEJB (
PLAYEREJB_PLAYERID varchar2(256) null,
 TEAMEJB_TEAMID varchar2(256) null
)
alter table TEAMEJB
add constraint FK_LEAGUE foreign key (LEAGUEEJB_LEAGUEID)
references LEAGUEEJB (LEAGUEID)
alter table PLAYEREJBTEAMEJB
add constraint FK_TEAMS foreign key (PLAYEREJB_PLAYERID)
references PLAYEREJB (PLAYERID)
alter table PLAYEREJBTEAMEJB
add constraint FK_PLAYERS foreign key (TEAMEJB_TEAMID)
references TEAMEJB (TEAMID)
```

Here is a corresponding sample `sun-cmp-mappings.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<sun-cmp-mappings>
<sun-cmp-mapping>
```

```
<schema>Roster</schema>
<entity-mapping>
    <ejb-name>TeamEJB</ejb-name>
    <table-name>TEAMEJB</table-name>
    <cmp-field-mapping>
        <field-name>teamId</field-name>
        <column-name>TEAMEJB.TEAMID</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>name</field-name>
        <column-name>TEAMEJB.NAME</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>city</field-name>
        <column-name>TEAMEJB.CITY</column-name>
    </cmp-field-mapping>
    <cmr-field-mapping>
        <cmr-field-name>league</cmr-field-name>
        <column-pair>
            <column-name>TEAMEJB.LEAGUEEJB_LEAGUEID</column-name>
            <column-name>LEAGUEEJB.LEAGUEID</column-name>
        </column-pair>
        <fetched-with>
            <none/>
        </fetched-with>
    </cmr-field-mapping>
    <cmr-field-mapping>
        <cmr-field-name>players</cmr-field-name>
        <column-pair>
            <column-name>TEAMEJB.TEAMID</column-name>
            <column-name>PLAYEREJBTEAMEJB.TEAMEJB_TEAMID</column-name>
        </column-pair>
        <column-pair>
            <column-name>PLAYEREJBTEAMEJB.PLAYEREJB_PLAYERID</column-name>
            <column-name>PLAYEREJB.PLAYERID</column-name>
        </column-pair>
        <fetched-with>
            <none/>
        </fetched-with>
    </cmr-field-mapping>
</entity-mapping>
<entity-mapping>
    <ejb-name>PlayerEJB</ejb-name>
    <table-name>PLAYEREJB</table-name>
    <cmp-field-mapping>
        <field-name>position</field-name>
        <column-name>PLAYEREJB.POSITION</column-name>
    </cmp-field-mapping>
```

```
        <cmp-field-mapping>
            <field-name>playerId</field-name>
            <column-name>PLAYEREJB.PLAYERID</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>name</field-name>
            <column-name>PLAYEREJB.NAME</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>salary</field-name>
            <column-name>PLAYEREJB.SALARY</column-name>
        </cmp-field-mapping>
        <cmr-field-mapping>
            <cmr-field-name>teams</cmr-field-name>
            <column-pair>
                <column-name>PLAYEREJB.PLAYERID</column-name>
                <column-name>PLAYEREJBTEAMEJB.PLAYEREJB_PLAYERID</column-name>
            </column-pair>
            <column-pair>
                <column-name>PLAYEREJBTEAMEJB.TEAMEJB_TEAMID</column-name>
                <column-name>TEAMEJB.TEAMID</column-name>
            </column-pair>
            <fetched-with>
                <none/>
            </fetched-with>
        </cmr-field-mapping>
    </entity-mapping>
    <entity-mapping>
        <ejb-name>LeagueEJB</ejb-name>
        <table-name>LEAGUEEJB</table-name>
        <cmp-field-mapping>
            <field-name>leagueId</field-name>
            <column-name>LEAGUEEJB.LEAGUEID</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>name</field-name>
            <column-name>LEAGUEEJB.NAME</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>sport</field-name>
            <column-name>LEAGUEEJB.SPORT</column-name>
        </cmp-field-mapping>
        <cmr-field-mapping>
            <cmr-field-name>teams</cmr-field-name>
            <column-pair>
                <column-name>LEAGUEEJB.LEAGUEID</column-name>
                <column-name>TEAMEJB.LEAGUEEJB_LEAGUEID</column-name>
            </column-pair>
```

```
            <fetched-with>
                <none/>
            </fetched-with>
        </cmr-field-mapping>
    </entity-mapping>
</sun-cmp-mapping>
</sun-cmp-mappings>
```

The sun-application-client.xml file

The element hierarchy in the `sun-application-client.xml` file is as follows:

```
sun-application-client
.  ejb-ref
.  .  ejb-ref-name
.  .  jndi-name
.  resource-ref
.  .  res-ref-name
.  .  jndi-name
.  .  default-resource-principal
.  .  .  name
.  .  .  password
.  resource-env-ref
.  .  resource-env-ref-name
.  .  jndi-name
.  service-ref
.  .  service-ref-name
.  .  port-info
.  .  .  service-endpoint-interface
.  .  .  wsdl-port
.  .  .  .  namespaceURI
.  .  .  .  localpart
.  .  .  stub-property
.  .  .  .  name
.  .  .  .  value
.  .  .  call-property
.  .  .  .  name
.  .  .  .  value
.  .  .  message-security-binding
.  .  .  .  message-security
.  .  .  .  .  message
.  .  .  .  .  .  java-method
.  .  .  .  .  .  .  method-name
.  .  .  .  .  .  .  method-params
.  .  .  .  .  .  .  .  method-param
.  .  .  .  .  .  operation-name
```

```
. . . . . request-protection
. . . . . response-protection
. . call-property
. . . name
. . . value
. . wsdl-override
. . service-impl-class
. . service-qname
. . . namespaceURI
. . . localpart
. message-destination
. . message-destination-name
. . jndi-name
```

The sun-acc.xml File

The element hierarchy in the `sun-acc.xml` file is as follows:

```
client-container
. target-server
. . description
. . security
. . . ssl
. . . cert-db
. auth-realm
. . property (with attributes)
. client-credential
. . property (with attributes)
. log-service
. . property (with attributes)
. message-security-config
. . provider-config
. . . request-policy
. . . response-policy
. . . property (with attributes)
. property (with attributes)
```

Alphabetical Listing of All Elements

"A" on page 299 "B" on page 302 "C" on page 304 "D" on page 324 "E" on page 327 "F" on page 335 "G" on page 337 "H" on page 339 "I" on page 339 "J" on page 341 "K" on page 345 "L" on page 346 "M" on page 351 "N" on page 361 "O" on page 362 "P" on page 363 "Q" on page 371 "R" on page 373 "S" on page 382 "T" on page 401 "U" on page 404 "V" on page 405 "W" on page 407

A

activation-config

Specifies an activation configuration, which includes the runtime configuration properties of the message-driven bean in its operational environment. For example, this can include information about the name of a physical JMS destination. Matches and overrides the `activation-config` element in the `ejb-jar.xml` file.

Superelements

"mdb-resource-adapter" on page 354 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–2 `activation-config` subelements

Element	Required	Description
"description" on page 326	zero or one	Specifies a text description of the activation configuration.
"activation-config-property" on page 299	one or more	Specifies an activation configuration property.

activation-config-property

Specifies the name and value of an activation configuration property.

Superelements

"activation-config" on page 299 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–3 `activation-config-property` subelements

Element	Required	Description
"activation-config-property-name" on page 300	only one	Specifies the name of an activation configuration property.
"activation-config-property-value" on page 300	only one	Specifies the value of an activation configuration property.

activation-config-property-name

Specifies the name of an activation configuration property.

Superelements

"activation-config-property" on page 299 (`sun-ejb-jar.xml`)

Subelements

none - contains data

activation-config-property-value

Specifies the value of an activation configuration property.

Superelements

"activation-config-property" on page 299 (`sun-ejb-jar.xml`)

Subelements

none - contains data

as-context

Specifies the authentication mechanism used to authenticate the client.

Superelements

"ior-security-config" on page 339 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `as-context` element.

TABLE A–4 `as-context` Subelements

Element	Required	Description
"auth-method" on page 301	only one	Specifies the authentication method. The only supported value is `USERNAME_PASSWORD`.
"realm" on page 373	only one	Specifies the realm in which the user is authenticated.
"required" on page 377	only one	Specifies whether the authentication method specified must be used for client authentication.

auth-method

Specifies the authentication method.

If the parent element is "as-context" on page 300, the only supported value is `USERNAME_PASSWORD`.

If the parent element is "login-config" on page 351, specifies the authentication mechanism for the web service endpoint. As a prerequisite to gaining access to any web resources protected by an authorization constraint, a user must be authenticated using the configured mechanism.

Superelements

"login-config" on page 351 (`sun-web.xml`), "as-context" on page 300 (`sun-ejb-jar.xml`)

Subelements

none - contains data

auth-realm

JAAS is available on the ACC. Defines the optional configuration for a JAAS authentication realm. Authentication realms require provider-specific properties, which vary depending on what a particular implementation needs. For more information about how to define realms, see "Realm Configuration" on page 50.

Superelements

"client-container" on page 313 (`sun-acc.xml`)

Subelements

The following table describes subelements for the `auth-realm` element.

TABLE A–5 `auth-realm` subelement

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `auth-realm` element.

TABLE A–6 `auth-realm` attributes

Attribute	Default	Description
`name`	none	Defines the name of this realm.
`classname`	none	Defines the Java class which implements this realm.

Example

Here is an example of the default file realm:

```
<auth-realm name="file"
   classname="com.sun.enterprise.security.auth.realm.file.FileRealm">
   <property name="file" value="domain-dir/config/keyfile"/>
   <property name="jaas-context" value="fileRealm"/>
</auth-realm>
```

Which properties an `auth-realm` element uses depends on the value of the `auth-realm` element's name attribute. The file realm uses `file` and `jaas-context` properties. Other realms use different properties. See "Realm Configuration" on page 50.

B

bean-cache

Specifies the entity bean cache properties. Used for entity beans and stateful session beans.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–7 `bean-cache` Subelements

Element	Required	Description
"max-cache-size" on page 353	zero or one	Specifies the maximum number of beans allowable in cache.
"is-cache-overflow-allowed" on page 340	zero or one	Deprecated.
"cache-idle-timeout-in-seconds" on page 307	zero or one	Specifies the maximum time that a stateful session bean or entity bean is allowed to be idle in cache before being passivated. Default value is 10 minutes (600 seconds).
"removal-timeout-in-seconds" on page 374	zero or one	Specifies the amount of time a bean remains before being removed. If `removal-timeout-in-seconds` is less than `idle-timeout`, the bean is removed without being passivated.
"resize-quantity" on page 377	zero or one	Specifies the number of beans to be created if the pool is empty (subject to the `max-pool-size` limit). Values are from 0 to MAX_INTEGER.
"victim-selection-policy" on page 406	zero or one	Specifies the algorithm that must be used by the container to pick victims. Applies only to stateful session beans.

Example

```
<bean-cache>
   <max-cache-size>100</max-cache-size>
   <cache-resize-quantity>10</cache-resize-quantity>
   <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
   <victim-selection-policy>LRU</victim-selection-policy>
      <cache-idle-timeout-in-seconds>600</cache-idle-timeout-in-seconds>
   <removal-timeout-in-seconds>5400</removal-timeout-in-seconds>
</bean-cache>
```

bean-pool

Specifies the pool properties of stateless session beans, entity beans, and message-driven bean.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `bean-pool` element.

TABLE A–8 `bean-pool` Subelements

Element	Required	Description
"steady-pool-size" on page 393	zero or one	Specifies the initial and minimum number of beans maintained in the pool. Default is 32.
"resize-quantity" on page 377	zero or one	Specifies the number of beans to be created if the pool is empty (subject to the `max-pool-size` limit). Values are from 0 to MAX_INTEGER.
"max-pool-size" on page 353	zero or one	Specifies the maximum number of beans in the pool. Values are from 0 to MAX_INTEGER. Default is to the EJB container value or 60.
"max-wait-time-in-millis" on page 354	zero or one	Deprecated.
"pool-idle-timeout-in-seconds" on page 365	zero or one	Specifies the maximum time that a bean is allowed to be idle in the pool. After this time, the bean is removed. This is a hint to the server. Default time is 600 seconds (10 minutes).

Example

```
<bean-pool>
   <steady-pool-size>10</steady-pool-size>
   <resize-quantity>10</resize-quantity>
   <max-pool-size>100</max-pool-size>
   <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
</bean-pool>
```

C

cache

Configures caching for web application components.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–9 `cache` Subelements

Element	Required	Description
"cache-helper" on page 306	zero or more	Specifies a custom class that implements the `CacheHelper` interface.
"default-helper" on page 324	zero or one	Allows you to change the properties of the default, built-in "cache-helper" on page 306 class.
"property (with attributes)" on page 368	zero or more	Specifies a cache property, which has a name and a value.
"cache-mapping" on page 308	zero or more	Maps a URL pattern or a servlet name to its cacheability constraints.

Attributes

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

TABLE A–10 `cache` Attributes

Attribute	Default	Description
`max-entries`	`4096`	(optional) Specifies the maximum number of entries the cache can contain. Must be a positive integer.
`timeout-in-seconds`	`30`	(optional) Specifies the maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. Can be overridden by a "timeout" on page 402 element.
`enabled`	`true`	(optional) Determines whether servlet and JSP caching is enabled.

Properties

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

TABLE A–11 cache Properties

Property	Default	Description
cacheClassName	com.sun.appserv.web.cache.LruCache	Specifies the fully qualified name of the class that implements the cache functionality. See "Cache Class Names" on page 306 for possible values.
MultiLRUSegmentSize	4096	Specifies the number of entries in a segment of the cache table that should have its own LRU (least recently used) list. Applicable only if cacheClassName is set to com.sun.appserv.web.cache.MultiLruCache.
MaxSize	unlimited; Long.MAX_VALUE	Specifies an upper bound on the cache memory size in bytes (KB or MB units). Example values are 32 KB or 2 MB. Applicable only if cacheClassName is set to com.sun.appserv.web.cache.BoundedMultiLruCache.

Cache Class Names

The following table lists possible values of the cacheClassName property.

TABLE A–12 cacheClassName Values

Value	Description
com.sun.appserv.web.cache.LruCache	A bounded cache with an LRU (least recently used) cache replacement policy.
com.sun.appserv.web.cache.BaseCache	An unbounded cache suitable if the maximum number of entries is known.
com.sun.appserv.web.cache.MultiLruCache	A cache suitable for a large number of entries (>4096). Uses the MultiLRUSegmentSize property.
com.sun.appserv.web.cache.BoundedMultiLruCache	A cache suitable for limiting the cache size by memory rather than number of entries. Uses the MaxSize property.

cache-helper

Specifies a class that implements the com.sun.appserv.web.cache.CacheHelper interface.

Superelements

"cache" on page 304 (sun-web.xml)

Subelements

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

TABLE A–13 `cache-helper` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

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

TABLE A–14 `cache-helper` Attributes

Attribute	Default	Description
`name`	`default`	Specifies a unique name for the helper class, which is referenced in the "cache-mapping" on page 308 element.
`class-name`	none	Specifies the fully qualified class name of the cache helper, which must implement the `com.sun.appserv.web.CacheHelper` interface.

cache-helper-ref

Specifies the `name` of the "cache-helper" on page 306 used by the parent "cache-mapping" on page 308 element.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none - contains data

cache-idle-timeout-in-seconds

Specifies the maximum time that a bean can remain idle in the cache. After this amount of time, the container can passivate this bean. A value of `0` specifies that beans never become candidates for passivation. Default is 600.

Applies to stateful session beans and entity beans.

Superelements

"bean-cache" on page 302 (`sun-ejb-jar.xml`)

Subelements

none - contains data

cache-mapping

Maps a URL pattern or a servlet name to its cacheability constraints.

Superelements

"cache" on page 304 (`sun-web.xml`)

Subelements

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

TABLE A–15 `cache-mapping` Subelements

Element	Required	Description
"servlet-name" on page 390	requires one `servlet-name` or `url-pattern`	Contains the name of a servlet.
"url-pattern" on page 405	requires one `servlet-name` or `url-pattern`	Contains a servlet URL pattern for which caching is enabled.
"cache-helper-ref" on page 307	required if `dispatcher`, `timeout`, `refresh-field`, `http-method`, `key-field`, and `constraint-field` are not used	Contains the `name` of the "cache-helper" on page 306 used by the parent `cache-mapping` element.
"dispatcher" on page 326	zero or one if `cache-helper-ref` is not used	Contains a comma-separated list of `RequestDispatcher` methods for which caching is enabled.
"timeout" on page 402	zero or one if `cache-helper-ref` is not used	Contains the "cache-mapping" on page 308 specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed.
"refresh-field" on page 373	zero or one if `cache-helper-ref` is not used	Specifies a field that gives the application component a programmatic way to refresh a cached entry.

TABLE A–15 `cache-mapping` Subelements *(Continued)*

Element	Required	Description
"http-method" on page 339	zero or more if `cache-helper-ref` is not used	Contains an HTTP method that is eligible for caching.
"key-field" on page 345	zero or more if `cache-helper-ref` is not used	Specifies a component of the key used to look up and extract cache entries.
"constraint-field" on page 320	zero or more if `cache-helper-ref` is not used	Specifies a cacheability constraint for the given `url-pattern` or `servlet-name`.

call-property

Specifies JAX-RPC property values that can be set on a `javax.xml.rpc.Call` object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC `Call` implementation.

Superelements

"port-info" on page 366, "service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

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

TABLE A–16 `call-property` subelements

Element	Required	Description
"name" on page 361	only one	Specifies the name of the entity.
"value" on page 405	only one	Specifies the value of the entity.

caller-propagation

Specifies whether the target accepts propagated caller identities. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

"sas-context" on page 382 (`sun-ejb-jar.xml`)

Subelements

none - contains data

cert-db

Not implemented. Included for backward compatibility only. Attribute values are ignored.

Superelements

"security" on page 385 (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `cert-db` element.

TABLE A–17 `cert-db` attributes

Attribute	Default	Description
`path`	none	Specifies the absolute path of the certificate database.
`password`	none	Specifies the password to access the certificate database.

check-all-at-commit

This element is not implemented. Do not use.

Superelements

"consistency" on page 320 (`sun-cmp-mappings.xml`)

check-modified-at-commit

Checks concurrent modification of fields in modified beans at commit time.

Superelements

"consistency" on page 320 (`sun-cmp-mappings.xml`)

Subelements

none - element is present or absent

check-version-of-accessed-instances

Checks the version column of the modified beans.

Version consistency allows the bean state to be cached between transactions instead of read from a database. The bean state is verified by primary key and version column values. This occurs during a custom query (for dirty instances only) or commit (for both clean and dirty instances).

The version column must be a numeric type, and must be in the primary table. You must provide appropriate update triggers for this column.

Superelements

"consistency" on page 320 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `check-version-of-accessed-instances` element.

TABLE A–18 `check-version-of-accessed-instances` Subelements

Element	Required	Description
"column-name" on page 318	only one	Specifies the name of the version column.

checkpoint-at-end-of-method

Enterprise Edition only. Do not use.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

checkpointed-methods

Enterprise Edition only. Do not use.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

class-loader

Configures the class loader for the web module.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–19 `class-loader` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

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

TABLE A–20 `class-loader` Attributes

Attribute	Default	Description
`extra-class-path`	null	(optional) Specifies additional classpath settings for this web module.
`delegate`	`true`	(optional) If `true`, the web module follows the standard class loader delegation model and delegates to its parent class loader first before looking in the local class loader. You must set this to `true` for a web application that accesses EJB components or that acts as a web service client or endpoint. If `false`, the web module follows the delegation model specified in the Servlet specification and looks in its class loader before looking in the parent class loader. It's safe to set this to `false` only for a web module that does not interact with any other modules.
`dynamic-reload-interval`		(optional) Not implemented. Included for backward compatibility with previous Sun Java System Web Server versions.

Note – If the `delegate` element is set to `false`, the class loader delegation behavior complies with the Servlet 2.4 specification, section 9.7.2. If set to its default value of `true`, classes and resources residing in container-wide library JAR files are loaded in preference to classes and resources packaged within the WAR file.

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

client-container

Defines the Application Server specific configuration for the application client container. This is the root element; there can only be one `client-container` element in a `sun-acc.xml` file. See "The sun-acc.xml File" on page 298.

Superelements

none

Subelements

The following table describes subelements for the `client-container` element.

TABLE A–21 `client-container` Subelements

Element	Required	Description
"target-server" on page 401	only one	Specifies the IIOP listener configuration of the target server.
"auth-realm" on page 301	zero or one	Specifies the optional configuration for JAAS authentication realm.
"client-credential" on page 314	zero or one	Specifies the default client credential that is sent to the server.
"log-service" on page 350	zero or one	Specifies the default log file and the severity level of the message.
"message-security-config" on page 358	zero or more	Specifies configurations for message security providers.
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `client-container` element.

TABLE A–22 `client-container` Attributes

Attribute	Default	Description
`send-password`	`true`	If `true`, specifies that client authentication credentials must be sent to the server. Without authentication credentials, all access to protected EJB components results in exceptions.

client-credential

Default client credentials that are sent to the server. If this element is present, the credentials are automatically sent to the server, without prompting the user for the user name and password on the client side.

Superelements

"client-container" on page 313 (`sun-acc.xml`)

Subelements

The following table describes subelements for the `client-credential` element.

TABLE A–23 `client-credential` subelement

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `client-credential` element.

TABLE A–24 `client-credential` attributes

Attribute	Default	Description
`user-name`	none	The user name used to authenticate the Application client container.
`password`	none	The password used to authenticate the Application client container.
`realm`	the default realm for the domain	(optional) The realm (specified by name) where credentials are to be resolved.

cmp

Describes runtime information for a CMP entity bean object for EJB 1.1 and EJB 2.1 beans.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–25 `cmp` Subelements

Element	Required	Description
"mapping-properties" on page 353	zero or one	This element is not implemented.
"is-one-one-cmp" on page 340	zero or one	This element is not implemented.
"one-one-finders" on page 362	zero or one	Describes the finders for CMP 1.1 beans.
"prefetch-disabled" on page 367	zero or one	Disables prefetching of entity bean states for the specified query methods.

cmp-field-mapping

The `cmp-field-mapping` element associates a field with one or more columns to which it maps. The column can be from a bean's primary table or any defined secondary table. If a field is mapped to multiple columns, the column listed first in this element is used as a source for getting the value from the database. The columns are updated in the order they appear. There is one `cmp-field-mapping` element for each `cmp-field` element defined in the `ejb-jar.xml` file.

Superelements

"entity-mapping" on page 334 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `cmp-field-mapping` element.

TABLE A–26 `cmp-field-mapping` Subelements

Element	Required	Description
"field-name" on page 336	only one	Specifies the Java identifier of a field. This identifier must match the value of the `field-name` subelement of the `cmp-field` that is being mapped.
"column-name" on page 318	one or more	Specifies the name of a column from the primary table, or the qualified table name (TABLE.COLUMN) of a column from a secondary or related table.
"read-only" on page 373	zero or one	Specifies that a field is read-only.
"fetched-with" on page 335	zero or one	Specifies the fetch group for this CMP field's mapping.

cmp-resource

Specifies the database to be used for storing CMP beans. For more information about this element, see "Configuring the CMP Resource" on page 177.

Superelements

"enterprise-beans" on page 332 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–27 `cmp-resource` Subelements

Element	Required	Description
"jndi-name" on page 342	only one	Specifies the absolute `jndi-name` of a JDBC resource or Persistence Manager resource.
"default-resource-principal" on page 325	zero or one	Specifies the default runtime bindings of a resource reference.
"property (with subelements)" on page 369	zero or more	Specifies a property name and value. Used to configure `PersistenceManagerFactory` properties if the `jndi-name` subelement refers to a JDBC resource.
"create-tables-at-deploy" on page 323	zero or one	If `true`, specifies that database tables are created for beans that are automatically mapped by the EJB container.

TABLE A–27 `cmp-resource` Subelements *(Continued)*

Element	Required	Description
"drop-tables-at-undeploy" on page 326	zero or one	If `true`, specifies that database tables that were automatically created when the bean(s) were last deployed are dropped when the bean(s) are undeployed.
"database-vendor-name" on page 324	zero or one	Specifies the name of the database vendor for which tables can be created.
"schema-generator-properties" on page 383	zero or one	Specifies field-specific type mappings and allows you to set the `use-unique-table-names` property.

cmr-field-mapping

A container-managed relationship field has a name and one or more column pairs that define the relationship. There is one `cmr-field-mapping` element for each `cmr-field` element in the `ejb-jar.xml` file. A relationship can also participate in a fetch group.

Superelements

"entity-mapping" on page 334 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `cmr-field-mapping` element.

TABLE A–28 `cmr-field-mapping` Subelements

Element	Required	Description
"cmr-field-name" on page 317	only one	Specifies the Java identifier of a field. Must match the value of the `cmr-field-name` subelement of the `cmr-field` that is being mapped.
"column-pair" on page 318	one or more	Specifies the pair of columns that determine the relationship between two database tables.
"fetched-with" on page 335	zero or one	Specifies the fetch group for this CMR field's relationship.

cmr-field-name

Specifies the Java identifier of a field. Must match the value of the `cmr-field-name` subelement of the `cmr-field` element in the `ejb-jar.xml` file.

Superelements

"cmr-field-mapping" on page 317 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

cmt-timeout-in-seconds

Overrides the Transaction Timeout setting of the Transaction Service for an individual bean. The default value, `0`, specifies that the default Transaction Service timeout is used. If positive, this value is used for all methods in the bean that start a new container-managed transaction. This value is *not* used if the bean joins a client transaction.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

column-name

Specifies the name of a column from the primary table, or the qualified table name (TABLE.COLUMN) of a column from a secondary or related table.

Superelements

"check-version-of-accessed-instances" on page 311, "cmp-field-mapping" on page 315, "column-pair" on page 318 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

column-pair

Specifies the pair of columns that determine the relationship between two database tables. Each `column-pair` must contain exactly two `column-name` subelements, which specify the column's names. The first `column-name` element names the table that this bean is mapped to, and the second `column-name` names the column in the related table.

Superelements

"cmr-field-mapping" on page 317, "secondary-table" on page 385 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `column-pair` element.

TABLE A–29 `column-pair` Subelements

Element	Required	Description
"column-name" on page 318	two	Specifies the name of a column from the primary table, or the qualified table name (TABLE.COLUMN) of a column from a secondary or related table.

commit-option

Specifies the commit option used on transaction completion. Valid values for the Application Server are `B` or `C`. Default value is `B`. Applies to entity beans.

Note – Commit option A is not supported for this Application Server release.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

confidentiality

Specifies if the target supports privacy-protected messages. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

"transport-config" on page 403 (`sun-ejb-jar.xml`)

Subelements

none - contains data

consistency

Specifies container behavior in guaranteeing transactional consistency of the data in the bean.

Superelements

"entity-mapping" on page 334 (`sun-cmp-mappings.xml`)

Subelements

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

TABLE A–30 `consistency` Subelements

Element	Required	Description
"none" on page 362	exactly one subelement is required	No consistency checking occurs.
"check-modified-at-commit" on page 310	exactly one subelement is required	Checks concurrent modification of fields in modified beans at commit time.
"lock-when-loaded" on page 349	exactly one subelement is required	Obtains an exclusive lock when the data is loaded.
"check-all-at-commit" on page 310		This element is not implemented. Do not use.
"lock-when-modified" on page 350		This element is not implemented. Do not use.
"check-version-of-accessed-instances" on page 311	exactly one subelement is required	Checks the version column of the modified beans.

constraint-field

Specifies a cacheability constraint for the given "url-pattern" on page 405 or "servlet-name" on page 390.

All `constraint-field` constraints must pass for a response to be cached. If there are `value` constraints, at least one of them must pass.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

The following table describes subelements for the `constraint-field` element.

TABLE A–31 `constraint-field` Subelements

Element	Required	Description
"constraint-field-value" on page 321	zero or more	Contains a value to be matched to the input parameter value.

Attributes

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

TABLE A–32 `constraint-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `request.attribute`, and `session.attribute`.
`cache-on-match`	`true`	(optional) If `true`, caches the response if matching succeeds. Overrides the same attribute in a "constraint-field-value" on page 321 subelement.
`cache-on-match-failure`	`false`	(optional) If `true`, caches the response if matching fails. Overrides the same attribute in a "constraint-field-value" on page 321 subelement.

constraint-field-value

Specifies a value to be matched to the input parameter value. The matching is case sensitive. For example:

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

Superelements

"constraint-field" on page 320 (`sun-web.xml`)

Subelements

none - contains data

Attributes

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

TABLE A–33 `constraint-field-value` Attributes

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

context-root

Contains the web context root for the application or web application. Overrides the corresponding element in the `application.xml` or `web.xml` file.

Superelements

"web" on page 407 (`sun-application.xml`), "sun-web-app" on page 398 (`sun-web.xml`)

Subelements

none - contains data

cookie-properties

Specifies session cookie properties.

Superelements

"session-config" on page 390 (`sun-web.xml`)

Subelements

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

TABLE A–34 `cookie-properties` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Properties

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

TABLE A–35 `cookie-properties` Properties

Property	Default	Description
`cookiePath`	Context path at which the web module is installed.	Specifies the pathname that is set when the cookie is created. The browser sends the cookie if the pathname for the request contains this pathname. If set to `/` (slash), the browser sends cookies to all URLs served by the Application Server. You can set the path to a narrower mapping to limit the request URLs to which the browser sends cookies.
`cookieMaxAgeSeconds`	`-1`	Specifies the expiration time (in seconds) after which the browser expires the cookie.
`cookieDomain`	(unset)	Specifies the domain for which the cookie is valid.
`cookieComment`	`Sun Java System Application Server Session Tracking Cookie`	Specifies the comment that identifies the session tracking cookie in the cookie file. Applications can provide a more specific comment for the cookie.

create-tables-at-deploy

Specifies whether database tables are created for beans that are automatically mapped by the EJB container. If `true`, creates tables in the database. If `false` (the default if this element is not present), does not create tables.

This element can be overridden during deployment. See Table 7–4.

Superelements

"cmp-resource" on page 316 (`sun-ejb-jar.xml`)

Subelements

none - contains data

D

database-vendor-name

Specifies the name of the database vendor for which tables can be created. Allowed values are `db2`, `mssql`, `oracle`, `derby`, and `sybase`, case-insensitive.

If no value is specified, a connection is made to the resource specified by the "jndi-name" on page 342 subelement of the "cmp-resource" on page 316 element, and the database vendor name is read. If the connection cannot be established, or if the value is not recognized, SQL-92 compliance is presumed.

This element can be overridden during deployment. See Table 7–4.

Superelements

"cmp-resource" on page 316 (`sun-ejb-jar.xml`)

Subelements

none - contains data

default

Specifies that a field belongs to the default hierarchical fetch group, and enables prefetching for a CMR field. To disable prefetching for specific query methods, use a "prefetch-disabled" on page 367 element in the `sun-ejb-jar.xml` file.

Superelements

"fetched-with" on page 335 (`sun-cmp-mappings.xml`)

Subelements

none - element is present or absent

default-helper

Passes property values to the built-in `default` "cache-helper" on page 306 class.

Superelements

"cache" on page 304 (`sun-web.xml`)

Subelements

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

TABLE A–36 `default-helper` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `default-helper` element.

TABLE A–37 `default-helper` Properties

Property	Default	Description
`cacheKeyGeneratorAttrName`	Uses the built-in `default` "cache-helper" on page 306 key generation, which concatenates the servlet path with "key-field" on page 345 values, if any.	The caching engine looks in the `ServletContext` for an attribute with a name equal to the value specified for this property to determine whether a customized `CacheKeyGenerator` implementation is used. An application can provide a customized key generator rather than using the `default` helper. See "CacheKeyGenerator Interface" on page 137.

default-resource-principal

Specifies the default principal (user) for the resource.

If this element is used in conjunction with a JMS Connection Factory resource, the `name` and `password` subelements must be valid entries in the Sun Java™ System Message Queue broker user repository. See the *Security Management* chapter in the *Sun Java System Message Queue 3.7 UR1 Administration Guide* for details.

Superelements

"resource-ref" on page 379 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "cmp-resource" on page 316, "mdb-connection-factory" on page 354 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–38 `default-resource-principal` Subelements

Element	Required	Description
"name" on page 361	only one	Specifies the default resource principal name used to sign on to a resource manager.
"password" on page 365	only one	Specifies password of the default resource principal.

description

Specifies a text description of the containing element.

Superelements

"property (with attributes)" on page 368 (`sun-web.xml`); "activation-config" on page 299, "method" on page 359 (`sun-ejb-jar.xml`); "target-server" on page 401 (`sun-acc.xml`)

Subelements

none - contains data

dispatcher

Specifies a comma-separated list of `RequestDispatcher` methods for which caching is enabled on the target resource. Valid values are `REQUEST`, `FORWARD`, `INCLUDE`, and `ERROR` . If this element is not specified, the default is `REQUEST`. See SRV.6.2.5 of the Servlet 2.4 specification for more information.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none - contains data

drop-tables-at-undeploy

Specifies whether database tables that were automatically created when the bean(s) were last deployed are dropped when the bean(s) are undeployed. If `true`, drops tables from the database. If `false` (the default if this element is not present), does not drop tables.

This element can be overridden during deployment. See Table 7–4.

Superelements

"cmp-resource" on page 316 (`sun-ejb-jar.xml`)

Subelements

none - contains data

E

ejb

Defines runtime properties for a single enterprise bean within the application. The subelements listed below apply to particular enterprise beans as follows:

- All types of beans: `ejb-name`, `ejb-ref`, `resource-ref`, `resource-env-ref`, `cmp`, `ior-security-config`, `gen-classes`, `jndi-name`, `use-thread-pool-id`
- Stateless session beans and message-driven beans: `bean-pool`
- Stateful session beans and entity beans: `bean-cache`
- Entity beans: `commit-option`, `bean-cache`, `bean-pool`, `is-read-only-bean`, `refresh-period-in-seconds`, `flush-at-end-of-method`
- Message-driven beans: `mdb-connection-factory`, `jms-durable-subscription-name`, `jms-max-messages-load`, `bean-pool`

Superelements

"enterprise-beans" on page 332 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–39 `ejb` Subelements

Element	Required	Description
"ejb-name" on page 330	only one	Matches the `ejb-name` in the corresponding `ejb-jar.xml` file.

TABLE A–39 `ejb` Subelements *(Continued)*

Element	Required	Description
"jndi-name" on page 342	zero or more	Specifies the absolute `jndi-name`.
"ejb-ref" on page 331	zero or more	Maps the absolute JNDI name to the `ejb-ref` element in the corresponding J2EE XML file.
"resource-ref" on page 379	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding J2EE XML file.
"resource-env-ref" on page 378	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding J2EE XML file.
"service-ref" on page 388	zero or more	Specifies runtime settings for a web service reference.
"pass-by-reference" on page 364	zero or one	Specifies the passing method used by an enterprise bean calling a remote interface method in another bean that is colocated within the same process.
"cmp" on page 315	zero or one	Specifies runtime information for a container-managed persistence (CMP) entity bean for EJB 1.1 and EJB 2.1 beans.
"principal" on page 367	zero or one	Specifies the principal (user) name in an enterprise bean that has the `run-as` role specified.
"mdb-connection-factory" on page 354	zero or one	Specifies the connection factory associated with a message-driven bean.
"jms-durable-subscription-name" on page 341	zero or one	Specifies the durable subscription associated with a message-driven bean.
"jms-max-messages-load" on page 341	zero or one	Specifies the maximum number of messages to load into a Java Message Service session at one time for a message-driven bean to serve. The default is 1.
"ior-security-config" on page 339	zero or one	Specifies the security information for the IOR.
"is-read-only-bean" on page 340	zero or one	Specifies that this entity bean is read-only.
"refresh-period-in-seconds" on page 374	zero or one	Specifies the rate at which a read-only-bean must be refreshed from the data source.

TABLE A–39 `ejb` Subelements *(Continued)*

Element	Required	Description
"commit-option" on page 319	zero or one	Has valid values of B or C. Default value is B.
"cmt-timeout-in-seconds" on page 318	zero or one	Overrides the Transaction Timeout setting of the Transaction Service for an individual bean.
"use-thread-pool-id" on page 405	zero or one	Specifies the thread pool from which threads are selected for remote invocations of this bean.
"gen-classes" on page 337	zero or one	Specifies all the generated class names for a bean.
"bean-pool" on page 303	zero or one `bean-pool`	Specifies the bean pool properties. Used for stateless session beans, entity beans, and message-driven bean pools.
"bean-cache" on page 302	zero or one `bean-pool`	Specifies the bean cache properties. Used only for stateful session beans and entity beans.
"mdb-resource-adapter" on page 354	zero or one	Specifies runtime configuration information for a message-driven bean.
"webservice-endpoint" on page 408	zero or more	Specifies information about a web service endpoint.
"flush-at-end-of-method" on page 337	zero or one	Specifies the methods that force a database flush after execution. Used for entity beans.
"checkpointed-methods" on page 311	zero or one	Enterprise Edition only. Do not use.
"checkpoint-at-end-of-method" on page 311	zero or one	Enterprise Edition only. Do not use.

Attributes

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

TABLE A–40 `ejb` Attributes

Attribute	Default	Description
`availability-enabled`	`false`	(optional)Enterprise Edition only. Do not use.

Example

```
<ejb>
   <ejb-name>CustomerEJB</ejb-name>
   <jndi-name>customer</jndi-name>
   <resource-ref>
      <res-ref-name>jdbc/SimpleBank</res-ref-name>
      <jndi-name>jdbc/__default</jndi-name>
   </resource-ref>
   <is-read-only-bean>false</is-read-only-bean>
   <commit-option>B</commit-option>
   <bean-pool>
      <steady-pool-size>10</steady-pool-size>
      <resize-quantity>10</resize-quantity>
      <max-pool-size>100</max-pool-size>
      <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
   </bean-pool>
   <bean-cache>
      <max-cache-size>100</max-cache-size>
      <resize-quantity>10</resize-quantity>
      <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
      <victim-selection-policy>LRU</victim-selection-policy>
   </bean-cache>
</ejb>
```

ejb-name

In the `sun-ejb-jar.xml` file, matches the `ejb-name` in the corresponding `ejb-jar.xml` file. The name must be unique among the names of the enterprise beans in the same EJB JAR file.

There is no architected relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the deployer assigns to the EJB component's home.

In the `sun-cmp-mappings.xml` file, specifies the `ejb-name` of the entity bean in the `ejb-jar.xml` file to which the container-managed persistence (CMP) bean corresponds.

Superelements

"ejb" on page 327, "method" on page 359 (`sun-ejb-jar.xml`); "entity-mapping" on page 334 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

ejb-ref

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

The `ejb-ref` element is used for the declaration of a reference to an EJB's home. Applies to session beans or entity beans.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`), "sun-application-client" on page 396 (`sun-application-client.xml`)

Subelements

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

TABLE A–41 `ejb-ref` Subelements

Element	Required	Description
"ejb-ref-name" on page 331	only one	Specifies the `ejb-ref-name` in the corresponding J2EE deployment descriptor file `ejb-ref` entry.
"jndi-name" on page 342	only one	Specifies the absolute `jndi-name` of a resource.

ejb-ref-name

Specifies the `ejb-ref-name` in the corresponding J2EE deployment descriptor file `ejb-ref` entry.

Superelements

"ejb-ref" on page 331 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

endpoint-address-uri

Specifies the relative path combined with the web server root to form the fully qualified endpoint address for a web service endpoint. This is a required element for EJB endpoints and an optional element for servlet endpoints.

For servlet endpoints, this value is relative to the web application context root. For EJB endpoints, the URI is relative to root of the web server (the first portion of the URI is a context root). The context root portion must not conflict with the context root of any web application deployed to the same web server.

In all cases, this value must be a fixed pattern (no ”*’ allowed).

If the web service endpoint is a servlet that implements only a single endpoint and has only one `url-pattern`, it is not necessary to set this value, because the web container derives it from the `web.xml` file.

Superelements

“webservice-endpoint” on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

Example

If the web server is listening at `http://localhost:8080`, the following `endpoint-address-uri`:

```
<endpoint-address-uri>StockQuoteService/StockQuotePort</endpoint-address-uri>
```

results in the following target endpoint address:

```
http://localhost:8080/StockQuoteService/StockQuotePort
```

enterprise-beans

Specifies all the runtime properties for an EJB JAR file in the application.

Superelements

“sun-ejb-jar” on page 398 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `enterprise-beans` element.

TABLE A–42 `enterprise-beans` Subelements

Element	Required	Description
"name" on page 361	zero or one	Specifies the name string.
"unique-id" on page 404	zero or one	Specifies a unique system identifier. This data is automatically generated and updated at deployment/redeployment. Do not specify or edit this value.
"ejb" on page 327	zero or more	Defines runtime properties for a single enterprise bean within the application.
"pm-descriptors" on page 365	zero or one	Deprecated.
"cmp-resource" on page 316	zero or one	Specifies the database to be used for storing container-managed persistence (CMP) beans in an EJB JAR file.
"message-destination" on page 355	zero or more	Specifies the name of a logical message destination.
"webservice-description" on page 407	zero or more	Specifies a name and optional publish location for a web service.

Example

```
<enterprise-beans>
 <ejb>
     <ejb-name>CustomerEJB</ejb-name>
     <jndi-name>customer</jndi-name>
     <resource-ref>
         <res-ref-name>jdbc/SimpleBank</res-ref-name>
         <jndi-name>jdbc/__default</jndi-name>
     </resource-ref>
     <is-read-only-bean>false</is-read-only-bean>
     <commit-option>B</commit-option>
     <bean-pool>
         <steady-pool-size>10</steady-pool-size>
        <resize-quantity>10</resize-quantity>
         <max-pool-size>100</max-pool-size>
         <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
     </bean-pool>
     <bean-cache>
         <max-cache-size>100</max-cache-size>
         <resize-quantity>10</resize-quantity>
         <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
         <victim-selection-policy>LRU</victim-selection-policy>
     </bean-cache>
```

```
 </ejb>
</enterprise-beans>
```

entity-mapping

Specifies the mapping a bean to database columns.

Superelements

"sun-cmp-mapping" on page 397 (`sun-cmp-mappings.xml`)

Subelements

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

TABLE A–43 `entity-mapping` Subelements

Element	Required	Description
"ejb-name" on page 330	only one	Specifies the name of the entity bean in the `ejb-jar.xml` file to which the CMP bean corresponds.
"table-name" on page 401	only one	Specifies the name of a database table. The table must be present in the database schema file.
"cmp-field-mapping" on page 315	one or more	Associates a field with one or more columns to which it maps.
"cmr-field-mapping" on page 317	zero or more	A container-managed relationship field has a name and one or more column pairs that define the relationship.
"secondary-table" on page 385	zero or more	Describes the relationship between a bean's primary and secondary table.
"consistency" on page 320	zero or one	Specifies container behavior in guaranteeing transactional consistency of the data in the bean.

establish-trust-in-client

Specifies if the target is capable of authenticating a client. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

"transport-config" on page 403 (`sun-ejb-jar.xml`)

Subelements

none - contains data

establish-trust-in-target

Specifies if the target is capable of authenticating *to* a client. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

"transport-config" on page 403 (`sun-ejb-jar.xml`)

Subelements

none - contains data

F

fetched-with

Specifies the fetch group configuration for fields and relationships. The `fetched-with` element has different allowed and default subelements based on its parent element and the data types of the fields.

- If there is no `fetched-with` subelement of a "cmp-field-mapping" on page 315, and the data type is *not* BLOB, CLOB, VARBINARY, LONGVARBINARY, or OTHER, `fetched-with` can have any valid subelement. The default subelement is as follows:

  ```
  <fetched-with><default/></fetched-with>
  ```

- If there is no `fetched-with` subelement of a "cmp-field-mapping" on page 315, and the data type is BLOB, CLOB, VARBINARY, LONGVARBINARY, or OTHER, `fetched-with` can have any valid subelement *except* `<default/>`. The default subelement is as follows:

  ```
  <fetched-with><none/></fetched-with>
  ```

- If there is no `fetched-with` subelement of a "cmr-field-mapping" on page 317, `fetched-with` can have any valid subelement. The default subelement is as follows:

  ```
  <fetched-with><none/></fetched-with>
  ```

Managed fields are multiple CMP or CMR fields that are mapped to the same column. A managed field can have any `fetched-with` subelement except `<default/>`. For additional information, see "Managed Fields" on page 168.

Superelements

"cmp-field-mapping" on page 315, "cmr-field-mapping" on page 317 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `fetched-with` element.

TABLE A–44 `fetched-with` Subelements

Element	Required	Description
"default" on page 324	exactly one subelement is required	Specifies that a CMP field belongs to the default hierarchical fetch group, which means it is fetched any time the bean is loaded from a database. Enables prefetching of a CMR field.
"level" on page 346	exactly one subelement is required	Specifies the level number of a hierarchical fetch group.
"named-group" on page 361	exactly one subelement is required	Specifies the name of an independent fetch group.
"none" on page 362	exactly one subelement is required	Specifies that this field or relationship is placed into its own individual fetch group, which means it is loaded from a database the first time it is accessed in this transaction.

field-name

Specifies the Java identifier of a field. This identifier must match the value of the `field-name` subelement of the `cmp-field` element in the `ejb-jar.xml` file.

Superelements

"cmp-field-mapping" on page 315 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

finder

Describes the finders for CMP 1.1 with a method name and query.

Superelements

"one-one-finders" on page 362 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–45 `finder` Subelements

Element	Required	Description
"method-name" on page 360	only one	Specifies the method name for the finder.
"query-params" on page 372	zero or one	Specifies the query parameters for the CMP 1.1 finder.
"query-filter" on page 371	zero or one	Specifies the query filter for the CMP 1.1 finder.
"query-variables" on page 372	zero or one	Specifies variables in query expression for the CMP 1.1 finder.
"query-ordering" on page 372	zero or one	Specifies the query ordering for the CMP 1.1 finder.

flush-at-end-of-method

Specifies the methods that force a database flush after execution. Applicable to entity beans.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `flush-at-end-of-method` element.

TABLE A–46 `flush-at-end-of-method` Subelements

Element	Required	Description
"method" on page 359	one or more	Specifies a bean method.

G

gen-classes

Specifies all the generated class names for a bean.

Note – This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `gen-class` element.

TABLE A–47 `gen-classes` Subelements

Element	Required	Description
"remote-impl" on page 375	zero or one	Specifies the fully-qualified class name of the generated `EJBObject` impl class.
"local-impl" on page 347	zero or one	Specifies the fully-qualified class name of the generated `EJBLocalObject` impl class.
"remote-home-impl" on page 375	zero or one	Specifies the fully-qualified class name of the generated EJBHome impl class.
"local-home-impl" on page 346	zero or one	Specifies the fully-qualified class name of the generated `EJBLocalHome` impl class.

group-name

Specifies a group name in the current realm.

Superelements

"security-role-mapping" on page 386 (`sun-application.xml`, `sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

H

http-method

Specifies an HTTP method that is eligible for caching. The default is `GET`.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none - contains data

I

idempotent-url-pattern

Enterprise Edition only. Do not use.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

integrity

Specifies if the target supports integrity-protected messages. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

"transport-config" on page 403 (`sun-ejb-jar.xml`)

Subelements

none - contains data

ior-security-config

Specifies the security information for the input-output redirection (IOR).

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–48 `ior-security-config` Subelements

Element	Required	Description
"transport-config" on page 403	zero or one	Specifies the security information for transport.
"as-context" on page 300	zero or one	Specifies the authentication mechanism used to authenticate the client. If specified, it is `USERNAME_PASSWORD`.
"sas-context" on page 382	zero or one	Describes the sas-context fields.

is-cache-overflow-allowed

This element is deprecated. Do not use.

Superelements

"bean-cache" on page 302 (`sun-ejb-jar.xml`)

is-one-one-cmp

This element is not used.

Superelements

"cmp" on page 315 (`sun-ejb-jar.xml`)

is-read-only-bean

Specifies that this entity bean is a read-only bean if `true`. If this element is absent, the default value of `false` is used.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

J

java-method

Specifies a method.

Superelements

"message" on page 355 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

The following table describes subelements for the `java-method` element.

TABLE A–49 `java-method` Subelements

Element	Required	Description
"method-name" on page 360	only one	Specifies a method name.
"method-params" on page 360	zero or one	Specifies fully qualified Java type names of method parameters.

jms-durable-subscription-name

Specifies the durable subscription associated with a message-driven bean class. Only applies to the Java Message Service Topic Destination type, and only when the message-driven bean deployment descriptor subscription durability is Durable.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

jms-max-messages-load

Specifies the maximum number of messages to load into a Java Message Service session at one time for a message-driven bean to serve. The default is 1.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

jndi-name

Specifies the absolute `jndi-name` of a URL resource or a resource.

For entity beans and session beans, this value specifies the global JNDI name of the `EJBHome` object. It is only needed if the entity or session bean exposes a remote view.

For JMS message-driven beans, this is the JNDI name of the JMS resource from which the message-driven bean consumes JMS messages. This information is alternatively specified within the "activation-config" on page 299 subelement of the "mdb-resource-adapter" on page 354 element. For more information about JMS resources, see Chapter 14, "Using the Java Message Service."

Superelements

"ejb-ref" on page 331, "message-destination" on page 355, "resource-env-ref" on page 378, "resource-ref" on page 379 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "cmp-resource" on page 316, "ejb" on page 327, "mdb-connection-factory" on page 354 (`sun-ejb-jar.xml`)

Subelements

none - contains data

jsp-config

Specifies JSP configuration information.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–50 `jsp-config` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property.

Properties

The default property values are tuned for development of JSP files at the cost of performance. To maximize performance, set `jsp-config` properties to these non-default values:

- `development` - `false` (as an alternative, set to `true` and give `modificationTestInterval` a large value)
- `mappedfile` - `false`
- `trimSpaces` - `true`
- `suppressSmap` - `true`
- `fork` - `false` (on Solaris)
- `classdebuginfo` - `false`

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

TABLE A–51 `jsp-config` Properties

Property	Default	Description
`checkInterval`	`0`	If `development` is set to `false` and `checkInterval` is greater than zero, background compilations are enabled. The `checkInterval` is the time in seconds between checks to see if a JSP file needs to be recompiled.
`classdebuginfo`	`true`	Specifies whether the generated Java servlets are compiled with the debug option set (`-g` for `javac`).
`classpath`	created dynamically based on the current web application	Specifies the classpath to use when compiling generated servlets.
`compiler`	`javac`	Specifies the compiler Ant uses to compile JSP files. See the Ant documentation for more information: http://antinstaller.sourceforge.net/manual/manual/

TABLE A–51 `jsp-config` Properties *(Continued)*

Property	Default	Description
`development`	`true`	If set to `true`, enables development mode, which allows JSP files to be checked for modification. Specify the frequency at which JSPs are checked using the `modificationTestInterval` property.
`dumpSmap`	`false`	If set to `true`, dumps SMAP information for JSR 45 debugging to a file. Set to `false` if `suppressSmap` is `true`.
`enablePooling`	`true`	If set to `true`, tag handler pooling is enabled.
`errorOnUseBeanInvalid ClassAttribute`	`false`	If set to `true`, issues an error when the value of the `class` attribute in a `useBean` action is not a valid bean class.
`fork`	`true`	Specifies that Ant forks the compiling of JSP files, using a JVM separate from the one in which Tomcat is running.
`genStrAsCharArray`	`false`	If set to `true`, generates text strings as `char` arrays, which improves performance in some cases.
`ieClassId`	`clsid:8AD9C840-044E -11D1-B3E9-00805F499D93`	Specifies the Java plug-in COM class ID for Internet Explorer. Used by the `<jsp:plugin>` tags.
`javaEncoding`	`UTF8`	Specifies the encoding for the generated Java servlet. This encoding is passed to the Java compiler that is used to compile the servlet as well. By default, the web container tries to use `UTF8`. If that fails, it tries to use the `javaEncoding` value. For encodings, see: http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html
`keepgenerated`	`true`	If set to `true`, keeps the generated Java files. If `false`, deletes the Java files.
`mappedfile`	`true`	If set to `true`, generates static content with one print statement per input line, to ease debugging.

TABLE A–51 `jsp-config` Properties *(Continued)*

Property	Default	Description
`modificationTestInterval`	`0`	Specifies the frequency in seconds at which JSPs are checked for modification. A value of `0` causes the JSP to be checked on every access. Used only if `development` is set to `true`.
`scratchdir`	The default work directory for the web application	Specifies the working directory created for storing all the generated code.
`suppressSmap`	`false`	If set to `true`, generation of SMAP information for JSR 45 debugging is suppressed.
`trimSpaces`	`false`	If set to `true`, trims white spaces in template text between actions or directives.
`usePrecompiled`	`false`	If set to `true`, an accessed JSP file is not compiled. Its precompiled servlet class is used instead. It is assumed that JSP files have been precompiled, and their corresponding servlet classes have been bundled in the web application's `WEB-INF/lib` or `WEB-INF/classes` directory.
`xpoweredBy`	`true`	If set to `true`, the X-Powered-By response header is added by the generated servlet.

K

key-field

Specifies a component of the key used to look up and extract cache entries. The web container looks for the named parameter, or field, in the specified scope.

If this element is not present, the web container uses the Servlet Path (the path section that corresponds to the servlet mapping that activated the current request). See the Servlet 2.4 specification, section SRV 4.4, for details on the Servlet Path.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none

Attributes

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

TABLE A–52 `key-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

L

level

Specifies the name of a hierarchical fetch group. The name must be an integer. Fields and relationships that belong to a hierarchical fetch group of equal (or lesser) value are fetched at the same time. The value of level must be greater than zero. Only one is allowed.

Superelements

"fetched-with" on page 335 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

local-home-impl

Specifies the fully-qualified class name of the generated `EJBLocalHome` `impl` class.

Note – This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

"gen-classes" on page 337 (`sun-ejb-jar.xml`)

Subelements

none - contains data

local-impl

Specifies the fully-qualified class name of the generated `EJBLocalObject impl` class.

Note – This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

"gen-classes" on page 337 (`sun-ejb-jar.xml`)

Subelements

none - contains data

locale-charset-info

Deprecated. For backward compatibility only. Use the "parameter-encoding" on page 363 subelement of "sun-web-app" on page 398 instead. Specifies information about the application's internationalization settings.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–53 `locale-charset-info` Subelements

Element	Required	Description
"locale-charset-map" on page 348	one or more	Maps a locale and an agent to a character encoding. Provided for backward compatibility. Used only for request processing, and only if no `parameter-encoding` is defined.
"parameter-encoding" on page 363	zero or one	Determines the default request character encoding and how the web container decodes parameters from forms according to a hidden field value.

Attributes

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

TABLE A–54 `locale-charset-info` Attributes

Attribute	Default	Description
`default-locale`	none	Although a value is required, the value is ignored. Use the `default-charset` attribute of the "parameter-encoding" on page 363 element.

locale-charset-map

Maps locales and agents to character encodings. Provided for backward compatibility. Used only for request processing. Used only if the character encoding is not specified in the request and cannot be derived from the optional "parameter-encoding" on page 363 element. For encodings, see `http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html`.

Superelements

"locale-charset-info" on page 347 (`sun-web.xml`)

Subelements

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

TABLE A–55 `locale-charset-map` Subelements

Element	Required	Description
"description" on page 326	zero or one	Specifies an optional text description of a mapping.

Attributes

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

TABLE A–56 `locale-charset-map` Attributes

Attribute	Default	Description
`locale`	none	Specifies the locale name.
`agent`	none	(optional) Specifies the type of client that interacts with the application server. For a given locale, different agents can have different preferred character encodings. The value of this attribute must exactly match the value of the `user-agent` HTTP request header sent by the client. See Table A–57 for more information.
`charset`	none	Specifies the character encoding to which the locale maps.

Example Agents

The following table specifies example `agent` attribute values.

TABLE A–57 Example `agent` Attribute Values

Agent	`user-agent` Header and `agent` Attribute Value
Internet Explorer 5.00 for Windows 2000	`Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)`
Netscape 4.7.7 for Windows 2000	`Mozilla/4.77 [en] (Windows NT 5.0; U)`
Netscape 4.7 for Solaris	`Mozilla/4.7 [en] (X11; u; Sun OS 5.6 sun4u)`

localpart

Specifies the local part of a QNAME.

Superelements

"service-qname" on page 387, "wsdl-port" on page 409 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

lock-when-loaded

Places a database update lock on the rows corresponding to the bean whenever the bean is loaded. How the lock is placed is database-dependent. The lock is released when the transaction finishes (commit or rollback). While the lock is in place, other database users have read access to the bean.

Superelements

"consistency" on page 320 (`sun-cmp-mappings.xml`)

Subelements

none - element is present or absent

lock-when-modified

This element is not implemented. Do not use.

Superelements

"consistency" on page 320 (`sun-cmp-mappings.xml`)

log-service

Specifies configuration settings for the log file.

Superelements

"client-container" on page 313 (`sun-acc.xml`)

Subelements

The following table describes subelements for the `log-service` element.

TABLE A–58 `log-service` subelement

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `log-service` element.

TABLE A–59 `log-service` attributes

Attribute	Default	Description
`log-file`	*your-ACC-dir*`/logs/client.log`	(optional) Specifies the file where the application client container logging information is stored.
`level`	`SEVERE`	(optional) Sets the base level of severity. Messages at or above this setting get logged to the log file.

login-config

Specifies the authentication configuration for an EJB web service endpoint. Not needed for servlet web service endpoints. A servlet's security configuration is contained in the `web.xml` file.

Superelements

"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

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

TABLE A–60 `login-config` subelements

Element	Required	Description
"auth-method" on page 301	only one	Specifies the authentication method.

M

manager-properties

Specifies session manager properties.

Superelements

"session-manager" on page 390 (`sun-web.xml`)

Subelements

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

TABLE A–61 `manager-properties` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Properties

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

TABLE A–62 `manager-properties` Properties

Property	Default	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions. This is also the interval at which sessions are passivated if `maxSessions` is exceeded. To prevent data inconsistency, set this value lower than the frequency at which session data changes. For example, this value should be as low as possible (`1` second) for a hit counter servlet on a frequently accessed web site, or the last few hits might be lost each time the server is restarted. Applicable only if the `persistence-type` attribute of the parent "session-manager" on page 390 element is `file`.
`maxSessions`	`-1`	Specifies the maximum number of sessions that are permitted in the cache, or `-1` for no limit. After this, an attempt to create a new session causes an `IllegalStateException` to be thrown. The session manager passivates sessions to the persistent store when this maximum is reached. Applicable only if the `persistence-type` attribute of the parent "session-manager" on page 390 element is `file`.

TABLE A–62 `manager-properties` Properties *(Continued)*

Property	Default	Description
`sessionFilename`	none; state is not preserved across restarts	Specifies the absolute or relative path to the directory in which the session state is preserved between application restarts, if preserving the state is possible. A relative path is relative to the temporary directory for this web application. Applicable only if the `persistence-type` attribute of the parent "session-manager" on page 390 element is `memory`.

mapping-properties

This element is not implemented.

Superelements

"cmp" on page 315 (`sun-ejb-jar.xml`)

max-cache-size

Specifies the maximum number of beans allowable in cache. A value of zero indicates an unbounded cache. In reality, there is no hard limit. The max-cache-size limit is just a hint to the cache implementation. Default is 512.

Applies to stateful session beans and entity beans.

Superelements

"bean-cache" on page 302 (`sun-ejb-jar.xml`)

Subelements

none - contains data

max-pool-size

Specifies the maximum number of bean instances in the pool. Values are from 0 (1 for message-driven bean) to MAX_INTEGER. A value of 0 means the pool is unbounded. Default is 64.

Applies to all beans.

Superelements

"bean-pool" on page 303 (`sun-ejb-jar.xml`)

Subelements

none - contains data

max-wait-time-in-millis

This element is deprecated. Do not use.

Superelements

"bean-pool" on page 303 (`sun-ejb-jar.xml`)

mdb-connection-factory

Specifies the connection factory associated with a message-driven bean. Queue or Topic type must be consistent with the Java Message Service Destination type associated with the message-driven bean class.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `mdb-connection-factory` element.

TABLE A–63 `mdb-connection-factory` Subelements

Element	Required	Description
"jndi-name" on page 342	only one	Specifies the absolute `jndi-name`.
"default-resource-principal" on page 325	zero or one	Specifies the default sign-on (name/password) to the resource manager.

mdb-resource-adapter

Specifies runtime configuration information for a message-driven bean.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `mdb-resource-adapter` element.

TABLE A–64 `mdb-resource-adapter` subelements

Element	Required	Description
"resource-adapter-mid" on page 378	zero or one	Specifies a resource adapter module ID.
"activation-config" on page 299	one or more	Specifies an activation configuration.

message

Specifies the methods or operations to which message security requirements apply.

Superelements

"message-security" on page 356 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

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

TABLE A–65 `message` Subelements

Element	Required	Description
"java-method" on page 341	zero or one	Specifies the methods or operations to which message security requirements apply.
"operation-name" on page 362	zero or one	Specifies the WSDL name of an operation of a web service.

message-destination

Specifies the name of a logical `message-destination` defined within an application. The `message-destination-name` matches the corresponding `message-destination-name` in the corresponding J2EE deployment descriptor file.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "enterprise-beans" on page 332 (`sun-ejb-jar.xml`), "sun-application-client" on page 396 (`sun-application-client.xml`)

Subelements

The following table describes subelements for the `message-destination` element.

TABLE A–66 `message-destination` subelements

Element	Required	Description
"message-destination-name" on page 356	only one	Specifies the name of a logical message destination defined within the corresponding J2EE deployment descriptor file.
"jndi-name" on page 342	only one	Specifies the `jndi-name` of the associated entity.

message-destination-name

Specifies the name of a logical message destination defined within the corresponding J2EE deployment descriptor file.

Superelements

"message-destination" on page 355 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

message-security

Specifies message security requirements.

- If the grandparent element is "webservice-endpoint" on page 408, these requirements pertain to request and response messages of the endpoint.
- If the grandparent element is "port-info" on page 366, these requirements pertain to the port of the referenced service.

Superelements

"message-security-binding" on page 357 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

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

TABLE A–67 `message-security` Subelements

Element	Required	Description
"message" on page 355	one or more	Specifies the methods or operations to which message security requirements apply.
"request-protection" on page 376	zero or one	Defines the authentication policy requirements of the application's request processing.
"response-protection" on page 381	zero or one	Defines the authentication policy requirements of the application's response processing.

message-security-binding

Specifies a custom authentication provider binding for a parent "webservice-endpoint" on page 408 or "port-info" on page 366 element in one or both of these ways:

- By binding to a specific provider
- By specifying the message security requirements enforced by the provider

Superelements

"webservice-endpoint" on page 408, "port-info" on page 366 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

The following table describes subelements for the `message-security-binding` element.

TABLE A–68 `message-security-binding` Subelements

Element	Required	Description
"message-security" on page 356	zero or more	Specifies message security requirements.

Attributes

The following table describes attributes for the `message-security-binding` element.

TABLE A–69 `message-security-binding` Attributes

Attribute	Default	Description
`auth-layer`	none	Specifies the message layer at which authentication is performed. The value must be `SOAP`.

TABLE A–69 `message-security-binding` Attributes *(Continued)*

Attribute	Default	Description
`provider-id`	none	(optional) Specifies the authentication provider used to satisfy application-specific message security requirements. If this attribute is not specified, a default provider is used, if it is defined for the message layer. if no default provider is defined, authentication requirements defined in the `message-security-binding` are not enforced.

message-security-config

Specifies configurations for message security providers.

Superelements

"client-container" on page 313 (`sun-acc.xml`)

Subelements

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

TABLE A–70 `message-security-config` Subelements

Element	Required	Description
"provider-config" on page 370	one or more	Specifies a configuration for one message security provider.

Attributes

The following table describes attributes for the `message-security-config` element.

TABLE A–71 `message-security-config` Attributes

Attribute	Default	Description
`auth-layer`	none	Specifies the message layer at which authentication is performed. The value must be `SOAP`.
`default-provider`	none	(optional) Specifies the server provider that is invoked for any application not bound to a specific server provider.

TABLE A–71 `message-security-config` Attributes *(Continued)*

Attribute	Default	Description
`default-client-provider`	none	(optional) Specifies the client provider that is invoked for any application not bound to a specific client provider.

method

Specifies a bean method.

Superelements

"flush-at-end-of-method" on page 337 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–72 `method` Subelements

Element	Required	Description
"description" on page 326	zero or one	Specifies an optional text description.
"ejb-name" on page 330	zero or one	Matches the `ejb-name` in the corresponding `ejb-jar.xml` file.
"method-name" on page 360	only one	Specifies a method name.
"method-intf" on page 359	zero or one	Specifies the method interface to distinguish between methods with the same name in different interfaces.
"method-params" on page 360	zero or one	Specifies fully qualified Java type names of method parameters.

method-intf

Specifies the method interface to distinguish between methods with the same name in different interfaces. Allowed values are `Home`, `Remote`, `LocalHome`, and `Local`.

Superelements

"method" on page 359 (`sun-ejb-jar.xml`)

Subelements

none - contains data

method-name

Specifies a method name or * (an asterisk) for all methods. If a method is overloaded, specifies all methods with the same name.

Superelements

"java-method" on page 341 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "finder" on page 336, "query-method" on page 371 , "method" on page 359 (`sun-ejb-jar.xml`)

Subelements

none - contains data

Examples

```
<method-name>findTeammates</method-name>
```

```
<method-name>*</method-name>
```

method-param

Specifies the fully qualified Java type name of a method parameter.

Superelements

"method-params" on page 360 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

method-params

Specifies fully qualified Java type names of method parameters.

Superelements

"java-method" on page 341 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "query-method" on page 371, "method" on page 359 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `method-params` element.

TABLE A–73 `method-params` Subelements

Element	Required	Description
"method-param" on page 360	zero or more	Specifies the fully qualified Java type name of a method parameter.

N

name

Specifies the name of the entity.

Superelements

"call-property" on page 309, "default-resource-principal" on page 325, "stub-property" on page 394 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "enterprise-beans" on page 332, "principal" on page 367, "property (with subelements)" on page 369 (`sun-ejb-jar.xml`)

Subelements

none - contains data

named-group

Specifies the name of one independent fetch group. All the fields and relationships that are part of a named group are fetched at the same time. A field belongs to only one fetch group, regardless of what type of fetch group is used.

Superelements

"fetched-with" on page 335 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

namespaceURI

Specifies the namespace URI.

Superelements

"service-qname" on page 387, "wsdl-port" on page 409 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

none

Specifies that this field or relationship is fetched by itself, with no other fields or relationships.

Superelements

"consistency" on page 320, "fetched-with" on page 335 (`sun-cmp-mappings.xml`)

Subelements

none - element is present or absent

O

one-one-finders

Describes the finders for CMP 1.1 beans.

Superelements

"cmp" on page 315 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `one-one-finders` element.

TABLE A–74 `one-one-finders` Subelements

Element	Required	Description
"finder" on page 336	one or more	Describes the finders for CMP 1.1 with a method name and query.

operation-name

Specifies the WSDL name of an operation of a web service.

Superelements

"message" on page 355 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

P

parameter-encoding

Specifies the default request character encoding and how the web container decodes parameters from forms according to a hidden field value.

If both the "sun-web-app" on page 398 and "locale-charset-info" on page 347 elements have `parameter-encoding` subelements, the subelement of `sun-web-app` takes precedence. For encodings, see `http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html`.

Superelements

"locale-charset-info" on page 347, "sun-web-app" on page 398 (`sun-web.xml`)

Subelements

none

Attributes

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

TABLE A–75 `parameter-encoding` Attributes

Attribute	Default	Description
`form-hint-field`	none	(optional) The name of the hidden field in the form. This field specifies the character encoding the web container uses for `request.getParameter` and `request.getReader` calls when the charset is not set in the request's `content-type` header.
`default-charset`	`ISO-8859-1`	(optional) The default request character encoding.

pass-by-reference

Specifies the passing method used by a servlet or enterprise bean calling a remote interface method in another bean that is colocated within the same process.

- If `false` (the default if this element is not present), this application uses pass-by-value semantics.
- If `true`, this application uses pass-by-reference semantics.

Note – The `pass-by-reference` element only applies to remote calls. As defined in the EJB 2.1 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 passing semantics for calls to remote interfaces comply with the EJB 2.1 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's 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 element is set to `true`, parameters and return values should be considered read-only. The behavior of a program that modifies such parameters or return values is undefined.

When a servlet or enterprise bean calls a remote interface method in another bean that is colocated within the same process, by default the Application Server makes copies of all the call parameters in order to preserve the pass-by-value semantics. This increases the call overhead and decreases performance.

However, if the calling method does not change the object being passed as a parameter, it is safe to pass the object itself without making a copy of it. To do this, set the pass-by-reference value to `true`.

The setting of this element in the `sun-application.xml` file applies to all EJB modules in the application. For an individually deployed EJB module, you can set the same element in the `sun-ejb-jar.xml` file. If `pass-by-reference` is used at both the bean and application level, the bean level takes precedence.

Superelements

"sun-application" on page 395 (`sun-application.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

password

Specifies the password for the principal.

Superelements

"default-resource-principal" on page 325 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

pm-descriptors

This element and its subelements are deprecated. Do not use.

Superelements

"enterprise-beans" on page 332 (`sun-ejb-jar.xml`)

pool-idle-timeout-in-seconds

Specifies the maximum time, in seconds, that a bean instance is allowed to remain idle in the pool. When this timeout expires, the bean instance in a pool becomes a candidate for passivation or deletion. This is a hint to the server. A value of 0 specifies that idle beans remain in the pool indefinitely. Default value is 600.

Applies to stateless session beans, entity beans, and message-driven beans.

Note – For a stateless session bean or a message-driven bean, the bean is removed (garbage collected) when the timeout expires.

Superelements

"bean-pool" on page 303 (`sun-ejb-jar.xml`)

Subelements

none - contains data

port-component-name

Specifies a unique name for a port component within a web or EJB module.

Superelements

"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

port-info

Specifies information for a port within a web service reference.

Either a `service-endpoint-interface` or a `wsdl-port` or both must be specified. If both are specified, `wsdl-port` specifies the port that the container chooses for container-managed port selection.

The same `wsdl-port` value must not appear in more than one `port-info` element within the same `service-ref`.

If a `service-endpoint-interface` is using container-managed port selection, its value must not appear in more than one `port-info` element within the same `service-ref`.

Superelements

"service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

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

TABLE A–76 `port-info` subelements

Element	Required	Description
"service-endpoint-interface" on page 387	zero or one	Specifies the web service reference name relative to `java:comp/env`.
"wsdl-port" on page 409	zero or one	Specifies the WSDL port.
"stub-property" on page 394	zero or more	Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Stub` object before it is returned to the web service client.
"call-property" on page 309	zero or more	Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Call` object before it is returned to the web service client.

TABLE A–76 `port-info` subelements *(Continued)*

Element	Required	Description
"message-security-binding" on page 357	zero or one	Specifies a custom authentication provider binding.

prefetch-disabled

Disables prefetching of entity bean states for the specified query methods. Container-managed relationship fields are prefetched if their "fetched-with" on page 335 element is set to "default" on page 324.

Superelements

"cmp" on page 315 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `prefetch-disabled` element.

TABLE A–77 `prefetch-disabled` Subelements

Element	Required	Description
"query-method" on page 371	one or more	Specifies a query method.

principal

Defines a node that specifies a user name on the platform.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–78 `principal` Subelements

Element	Required	Description
"name" on page 361	only one	Specifies the name of the user.

principal-name

Contains the principal (user) name.

In an enterprise bean, specifies the principal (user) name that has the `run-as` role specified.

Superelements

"security-role-mapping" on page 386 (`sun-application.xml`, `sun-web.xml`, `sun-ejb-jar.xml`), "servlet" on page 389 (`sun-web.xml`)

Subelements

none - contains data

property (with attributes)

Specifies the name and value of a property. A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Application Server
- Needed by a system or object that Application Server doesn't have knowledge of, such as an LDAP server or a Java class

Superelements

"cache" on page 304, "cache-helper" on page 306, "class-loader" on page 311, "cookie-properties" on page 322, "default-helper" on page 324, "manager-properties" on page 351, "session-properties" on page 391, "store-properties" on page 393, "sun-web-app" on page 398 (`sun-web.xml`); "auth-realm" on page 301, "client-container" on page 313, "client-credential" on page 314, "log-service" on page 350, "provider-config" on page 370 (`sun-acc.xml`)

Subelements

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

TABLE A–79 `property` Subelements

Element	Required	Description
"description" on page 326	zero or one	Specifies an optional text description of a property.

Note – The `property` element in the `sun-acc.xml` file has no subelements.

Attributes

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

TABLE A–80 `property` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the property.
`value`	none	Specifies the value of the property.

Example

```
<property name="reapIntervalSeconds" value="20" />
```

property (with subelements)

Specifies the name and value of a property. A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Application Server
- Needed by a system or object that Application Server doesn't have knowledge of, such as an LDAP server or a Java class

Superelements

"cmp-resource" on page 316, "schema-generator-properties" on page 383 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–81 `property` subelements

Element	Required	Description
"name" on page 361	only one	Specifies the name of the property.
"value" on page 405	only one	Specifies the value of the property.

Example

```
<property>
   <name>use-unique-table-names</name>
   <value>true</value>
</property>
```

provider-config

Specifies a configuration for one message security provider.

Although the `request-policy` and `response-policy` subelements are optional, the `provider-config` element does nothing if they are not specified.

Use property subelements to configure provider-specific properties. Property values are passed to the provider when its `initialize` method is called.

Superelements

"message-security-config" on page 358 (`sun-acc.xml`)

Subelements

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

TABLE A–82 `provider-config` Subelements

Element	Required	Description
"request-policy" on page 375	zero or one	Defines the authentication policy requirements of the authentication provider's request processing.
"response-policy" on page 380	zero or one	Defines the authentication policy requirements of the authentication provider's response processing.
"property (with attributes)" on page 368	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `provider-config` element.

TABLE A–83 `provider-config` Attributes

Attribute	Default	Description
`provider-id`	none	Specifies the provider ID.

TABLE A–83 `provider-config` Attributes *(Continued)*

Attribute	Default	Description
`provider-type`	none	Specifies whether the provider is a `client`, `server`, or `client-server` authentication provider.
`class-name`	none	Specifies the Java implementation class of the provider. Client authentication providers must implement the `com.sun.enterprise.security.jauth.ClientAuthModule` interface. Server authentication providers must implement the `com.sun.enterprise.security.jauth.ServerAuthModule` interface. Client-server providers must implement both interfaces.

Q

query-filter

Specifies the query filter for the CMP 1.1 finder.

Superelements

"finder" on page 336 (`sun-ejb-jar.xml`)

Subelements

none - contains data

query-method

Specifies a query method.

Superelements

"prefetch-disabled" on page 367 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `query-method` element.

TABLE A–84 `query-method` Subelements

Element	Required	Description
"method-name" on page 360	only one	Specifies a method name.
"method-params" on page 360	only one	Specifies the fully qualified Java type names of method parameters.

query-ordering

Specifies the query ordering for the CMP 1.1 finder.

Superelements

"finder" on page 336 (`sun-ejb-jar.xml`)

Subelements

none - contains data

query-params

Specifies the query parameters for the CMP 1.1 finder.

Superelements

"finder" on page 336 (`sun-ejb-jar.xml`)

Subelements

none - contains data

query-variables

Specifies variables in the query expression for the CMP 1.1 finder.

Superelements

"finder" on page 336 (`sun-ejb-jar.xml`)

Subelements

none - contains data

R

read-only

Specifies that a field is read-only if `true`. If this element is absent, the default value is `false`.

Superelements

"cmp-field-mapping" on page 315 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

realm

Specifies the name of the realm used to process all authentication requests associated with this application. If this element is not specified or does not match the name of a configured realm, the default realm is used. For more information about realms, see "Realm Configuration" on page 50.

Superelements

"sun-application" on page 395 (`sun-application.xml`), "as-context" on page 300 (`sun-ejb-jar.xml`)

Subelements

none - contains data

refresh-field

Specifies a field that gives the application component a programmatic way to refresh a cached entry.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none

Attributes

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

TABLE A–85 `refresh-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

refresh-period-in-seconds

Specifies the rate at which a read-only-bean must be refreshed from the data source. If the value is less than or equal to zero, the bean is never refreshed; if the value is greater than zero, the bean instances are refreshed at the specified interval. This rate is just a hint to the container. Default is 0 (no refresh).

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

removal-timeout-in-seconds

Specifies the amount of time a bean instance can remain idle in the container before it is removed (timeout). A value of 0 specifies that the container does not remove inactive beans automatically. The default value is 5400.

If `removal-timeout-in-seconds` is less than or equal to `cache-idle-timeout-in-seconds`, beans are removed immediately without being passivated.

Applies to stateful session beans.

For related information, see "cache-idle-timeout-in-seconds" on page 307.

Superelements

"bean-cache" on page 302 (`sun-ejb-jar.xml`)

Subelements

none - contains data

remote-home-impl

Specifies the fully-qualified class name of the generated `EJBHome impl` class.

Note – This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

"gen-classes" on page 337 (`sun-ejb-jar.xml`)

Subelements

none - contains data

remote-impl

Specifies the fully-qualified class name of the generated `EJBObject impl` class.

Note – This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

"gen-classes" on page 337 (`sun-ejb-jar.xml`)

Subelements

none - contains data

request-policy

Defines the authentication policy requirements of the authentication provider's request processing.

Superelements

"provider-config" on page 370 (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `request-policy` element.

TABLE A–86 `request-policy` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

request-protection

Defines the authentication policy requirements of the application's request processing.

Superelements

"message-security" on page 356 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none

Attributes

The following table describes attributes for the `request-protection` element.

TABLE A–87 `request-protection` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).

TABLE A–87 request-protection Attributes *(Continued)*

Attribute	Default	Description
auth-recipient	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are before-content and after-content.

required

Specifies whether the authentication method specified must be used for client authentication. The value is true or false.

Superelements

"as-context" on page 300 (sun-ejb-jar.xml)

Subelements

none - contains data

res-ref-name

Specifies the res-ref-name in the corresponding J2EE deployment descriptor file resource-ref entry. The res-ref-name element specifies the name of a resource manager connection factory reference. The name must be unique within an enterprise bean.

Superelements

"resource-ref" on page 379 (sun-web.xml, sun-ejb-jar.xml, sun-application-client.xml)

Subelements

none - contains data

resize-quantity

Specifies the number of bean instances to be:

- Created, if a request arrives when the pool has less than "steady-pool-size" on page 393 quantity of beans (applies to pools only for creation). If the pool has more than steady-pool-size minus "resize-quantity" on page 377 of beans, then resize-quantity is still created.
- Removed, when the "pool-idle-timeout-in-seconds" on page 365 timer expires and a cleaner thread removes any unused instances.

- For caches, when "max-cache-size" on page 353 is reached, `resize-quantity` beans are selected for passivation using the "victim-selection-policy" on page 406. In addition, the "cache-idle-timeout-in-seconds" on page 307 or "removal-timeout-in-seconds" on page 374 timers passivate beans from the cache.
- For pools, when the "max-pool-size" on page 353 is reached, `resize-quantity` beans are selected for removal. In addition, the "pool-idle-timeout-in-seconds" on page 365 timer removes beans until `steady-pool-size` is reached.

Values are from 0 to MAX_INTEGER. The pool is not resized below the `steady-pool-size`. Default is 16.

Applies to stateless session beans, entity beans, and message-driven beans.

For EJB pools, the value can be defined in the EJB container. Default is 16.

For EJB caches, the value can be defined in the EJB container. Default is 32.

For message-driven beans, the value can be defined in the EJB container. Default is 2.

Superelements

"bean-cache" on page 302, "bean-pool" on page 303 (`sun-ejb-jar.xml`)

Subelements

none - contains data

resource-adapter-mid

Specifies the module ID of the resource adapter that is responsible for delivering messages to the message-driven bean.

Superelements

"mdb-resource-adapter" on page 354 (`sun-ejb-jar.xml`)

Subelements

none - contains data

resource-env-ref

Maps the `res-ref-name` in the corresponding J2EE deployment descriptor file `resource-env-ref` entry to the absolute `jndi-name` of a resource.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`), "sun-application-client" on page 396 (`sun-application-client.xml`)

Subelements

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

TABLE A–88 `resource-env-ref` Subelements

Element	Required	Description
"resource-env-ref-name" on page 379	only one	Specifies the `res-ref-name` in the corresponding J2EE deployment descriptor file `resource-env-ref` entry.
"jndi-name" on page 342	only one	Specifies the absolute `jndi-name` of a resource.

Example

```
<resource-env-ref>
   <resource-env-ref-name>jms/StockQueueName</resource-env-ref-name>
   <jndi-name>jms/StockQueue</jndi-name>
</resource-env-ref>
```

resource-env-ref-name

Specifies the `res-ref-name` in the corresponding J2EE deployment descriptor file `resource-env-ref` entry.

Superelements

"resource-env-ref" on page 378 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

resource-ref

Maps the `res-ref-name` in the corresponding J2EE deployment descriptor file `resource-ref` entry to the absolute `jndi-name` of a resource.

Note – Connections acquired from JMS connection factories are not shareable in the current release of the Application Server. The `res-sharing-scope` element in the `ejb-jar.xml` file `resource-ref` element is ignored for JMS connection factories.

When `resource-ref` specifies a JMS connection factory for the Sun Java System Message Queue, the `default-resource-principal` (name/password) must exist in the Message Queue user repository. Refer to the *Security Management* chapter in the *Sun Java System Message Queue 3.7 UR1 Administration Guide* for information on how to manage the Message Queue user repository.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`), "sun-application-client" on page 396 (`sun-application-client.xml`)

Subelements

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

TABLE A–89 `resource-ref` Subelements

Element	Required	Description
"res-ref-name" on page 377	only one	Specifies the `res-ref-name` in the corresponding J2EE deployment descriptor file `resource-ref` entry.
"jndi-name" on page 342	only one	Specifies the absolute `jndi-name` of a resource.
"default-resource-principal" on page 325	zero or one	Specifies the default principal (user) for the resource.

Example

```
<resource-ref>
   <res-ref-name>jdbc/EmployeeDBName</res-ref-name>
   <jndi-name>jdbc/EmployeeDB</jndi-name>
</resource-ref>
```

response-policy

Defines the authentication policy requirements of the authentication provider's response processing.

Superelements

"provider-config" on page 370 (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `response-policy` element.

TABLE A–90 `response-policy` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

response-protection

Defines the authentication policy requirements of the application's response processing.

Superelements

"message-security" on page 356 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none

Attributes

The following table describes attributes for the `response-protection` element.

TABLE A–91 `response-protection` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).

TABLE A–91 `response-protection` Attributes *(Continued)*

Attribute	Default	Description
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

role-name

Contains the `role-name` in the `security-role` element of the corresponding J2EE deployment descriptor file.

Superelements

"security-role-mapping" on page 386 (`sun-application.xml`, `sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

S

sas-context

Describes the sas-context fields.

Superelements

"ior-security-config" on page 339 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `sas-context` element.

TABLE A–92 `sas-context` Subelements

Element	Required	Description
"caller-propagation" on page 309	only one	Specifies whether the target accepts propagated caller identities. The values are NONE, SUPPORTED, or REQUIRED.

schema

Specifies the file that contains a description of the database schema to which the beans in this `sun-cmp-mappings.xml` file are mapped. If this element is empty, the database schema file is automatically generated at deployment time. Otherwise, the `schema` element names a `.dbschema` file with a pathname relative to the directory containing the `sun-cmp-mappings.xml` file, but without the `.dbschema` extension. See "Automatic Database Schema Capture" on page 176.

Superelements

"sun-cmp-mapping" on page 397 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

Examples

```
<schema/> <!-- use automatic schema generation -->

<schema>CompanySchema</schema> <!-- use "CompanySchema.dbschema" -->
```

schema-generator-properties

Specifies field-specific column attributes in `property` subelements.

Superelements

"cmp-resource" on page 316 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–93 `schema-generator-properties` Subelements

Element	Required	Description
"property (with subelements)" on page 369	zero or more	Specifies a property name and value.

Properties

The following table describes properties for the `schema-generator-properties` element.

TABLE A–94 `schema-generator-properties` Properties

Property	Default	Description
`use-unique-table-names`	`false`	Specifies that generated table names are unique within each application server domain. This property can be overridden during deployment. See Table 7–4.
bean-name.*field-name*.*attribute*	none	Defines a column attribute. For attribute descriptions, see Table A–95.

The following table lists the column attributes for properties defined in the `schema-generator-properties` element.

TABLE A–95 `schema-generator-properties` Column Attributes

Attribute	Description
`jdbc-type`	Specifies the JDBC type of the column created for the CMP field. The actual SQL type generated is based on this JDBC type but is database vendor specific.
`jdbc-maximum-length`	Specifies the maximum number of characters stored in the column corresponding to the CMP field. Applies only when the actual SQL that is generated for the column requires a length. For example, a `jdbc-maximum-length` of `32` on a CMP `String` field such as `firstName` normally results in a column definition such as VARCHAR(32). But if the `jdbc-type` is `CLOB` and you are deploying on Oracle, the resulting column definition is CLOB. No length is given, because in an Oracle database, a CLOB has no length.
`jdbc-precision`	Specifies the maximum number of digits stored in a column which represents a numeric type.
`jdbc-scale`	Specifies the number of digits stored to the right of the decimal point in a column that represents a floating point number.
`jdbc-nullable`	Specifies whether the column generated for the CMP field allows null values.

Example

```
<schema-generator-properties>
   <property>
         <name>Employee.firstName.jdbc-type</name>
         <value>char</value>
   </property>
```

```
    <property>
          <name>Employee.firstName.jdbc-maximum-length</name>
          <value>25</value>
    </property>
    <property>
          <name>use-unique-table-names</name>
          <value>true</value>
    </property>
</schema-generator-properties>
```

secondary-table

Specifies a bean's secondary table(s).

Superelements

"entity-mapping" on page 334 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `secondary-table` element.

TABLE A–96 `secondary table` Subelements

Element	Required	Description
"table-name" on page 401	only one	Specifies the name of a database table.
"column-pair" on page 318	one or more	Specifies the pair of columns that determine the relationship between two database tables.

security

Defines the SSL security configuration for IIOP/SSL communication with the target server.

Superelements

"target-server" on page 401 (`sun-acc.xml`)

Subelements

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

TABLE A–97 `security` Subelements

Element	Required	Description
"ssl" on page 392	only one	Specifies the SSL processing parameters.
"cert-db" on page 310	only one	Not implemented. Included for backward compatibility only.

security-role-mapping

Maps roles to users or groups in the currently active realm. See "Realm Configuration" on page 50.

The role mapping element maps a role, as specified in the EJB JAR `role-name` entries, to a environment-specific user or group. If it maps to a user, it must be a concrete user which exists in the current realm, who can log into the server using the current authentication method. If it maps to a group, the realm must support groups and the group must be a concrete group which exists in the current realm. To be useful, there must be at least one user in that realm who belongs to that group.

Superelements

"sun-application" on page 395 (`sun-application.xml`), "sun-web-app" on page 398 (`sun-web.xml`), "sun-ejb-jar" on page 398 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–98 `security-role-mapping` Subelements

Element	Required	Description
"role-name" on page 382	only one	Contains the `role-name` in the `security-role` element of the corresponding J2EE deployment descriptor file.
"principal-name" on page 368	one or more if no `group-name`, otherwise zero or more	Contains a principal (user) name in the current realm. In an enterprise bean, the principal must have the run-as role specified.
"group-name" on page 338	one or more if no `principal-name`, otherwise zero or more	Contains a group name in the current realm.

service-endpoint-interface

Specifies the web service reference name relative to `java:comp/env`.

Superelements

"port-info" on page 366 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

service-impl-class

Specifies the name of the generated service implementation class.

Superelements

"service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

service-qname

Specifies the WSDL service element that is being referred to.

Superelements

"service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`);
"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `service-qname` element.

TABLE A–99 `service-qname` subelements

Element	Required	Description
"namespaceURI" on page 361	only one	Specifies the namespace URI.

TABLE A–99 service-qname subelements *(Continued)*

Element	Required	Description
"localpart" on page 349	only one	Specifies the local part of a QNAME.

service-ref

Specifies runtime settings for a web service reference. Runtime information is only needed in the following cases:

- To define the port used to resolve a container-managed port
- To define the default Stub/Call property settings for Stub objects
- To define the URL of a final WSDL document to be used instead of the one associated with the `service-ref` in the standard J2EE deployment descriptor

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`), "sun-application-client" on page 396 (`sun-application-client.xml`)

Subelements

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

TABLE A–100 service-ref subelements

Element	Required	Description
"service-ref-name" on page 389	only one	Specifies the web service reference name relative to `java:comp/env`.
"port-info" on page 366	zero or more	Specifies information for a port within a web service reference.
"call-property" on page 309	zero or more	Specifies JAX-RPC property values that can be set on a `javax.xml.rpc.Call` object before it is returned to the web service client.
"wsdl-override" on page 409	zero or one	Specifies a valid URL pointing to a final WSDL document.
"service-impl-class" on page 387	zero or one	Specifies the name of the generated service implementation class.
"service-qname" on page 387	zero or one	Specifies the WSDL service element that is being referenced.

service-ref-name

Specifies the web service reference name relative to `java:comp/env`.

Superelements

"service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

servlet

Specifies a principal name for a servlet. Used for the `run-as` role defined in `web-xml`.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–101 `servlet` Subelements

Element	Required	Description
"servlet-name" on page 390	only one	Contains the name of a servlet, which is matched to a `servlet-name` in `web.xml`.
"principal-name" on page 368	zero or one	Contains a principal (user) name in the current realm.
"webservice-endpoint" on page 408	zero or more	Specifies information about a web service endpoint.

servlet-impl-class

Specifies the automatically generated name of the servlet implementation class.

Superelements

"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

servlet-name

Specifies the name of a servlet, which is matched to a `servlet-name` in `web.xml`. This name must be present in `web.xml`.

Superelements

"cache-mapping" on page 308, "servlet" on page 389 (`sun-web.xml`)

Subelements

none - contains data

session-config

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

Superelements

"sun-web-app" on page 398 (`sun-web.xml`)

Subelements

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

TABLE A–102 `session-config` Subelements

Element	Required	Description
"session-manager" on page 390	zero or one	Specifies session manager configuration information.
"session-properties" on page 391	zero or one	Specifies session properties.
"cookie-properties" on page 322	zero or one	Specifies session cookie properties.

session-manager

Specifies session manager information.

Superelements

"session-config" on page 390 (`sun-web.xml`)

Subelements

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

TABLE A–103 `session-manager` Subelements

Element	Required	Description
"manager-properties" on page 351	zero or one	Specifies session manager properties.
"store-properties" on page 393	zero or one	Specifies session persistence (storage) properties.

Attributes

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

TABLE A–104 `session-manager` Attributes

Attribute	Default	Description
`persistence-type`	`memory`	(optional) Specifies the session persistence mechanism. Allowed values are `memory` and `file`.

session-properties

Specifies session properties.

Superelements

"session-config" on page 390 (`sun-web.xml`)

Subelements

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

TABLE A–105 `session-properties` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Properties

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

TABLE A–106 `session-properties` Properties

Property	Default	Description
`timeoutSeconds`	`1800`	Specifies the default maximum inactive interval (in seconds) for all sessions created in this web module. If set to `0` or less, sessions in this web module never expire. If a `session-timeout` element is specified in the `web.xml` file, the `session-timeout` value overrides any `timeoutSeconds` value. If neither `session-timeout` nor `timeoutSeconds` is specified, the `timeoutSeconds` default is used. Note that the `session-timeout` element in `web.xml` is specified in minutes, not seconds.
`enableCookies`	`true`	Uses cookies for session tracking if set to `true`.
`enableURLRewriting`	`true`	Enables URL rewriting. This provides session tracking via URL rewriting when the browser does not accept cookies. You must also use an `encodeURL` or `encodeRedirectURL` call in the servlet or JSP.

ssl

Defines SSL processing parameters.

Superelements

"security" on page 385 (`sun-acc.xml`)

Subelements

none

Attributes

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

TABLE A–107 `ssl` attributes

Attribute	Default	Description
`cert-nickname`	none	(optional) The nickname of the server certificate in the certificate database or the PKCS#11 token. In the certificate, the name format is *tokenname*:*nickname*. Including the *tokenname*: part of the name in this attribute is optional.
`ssl2-enabled`	`false`	(optional) Determines whether SSL2 is enabled.
`ssl2-ciphers`	none	(optional) A space-separated list of the SSL2 ciphers used with the prefix + to enable or - to disable. For example, `+rc4`. Allowed values are `rc4, rc4export, rc2, rc2export, idea, des, desede3`.
`ssl3-enabled`	`true`	(optional) Determines whether SSL3 is enabled.
`ssl3-tls-ciphers`	none	(optional) A space-separated list of the SSL3 ciphers used, with the prefix + to enable or - to disable, for example `+rsa_des_sha`. Allowed SSL3 values are `rsa_rc4_128_md5, , rsa_des_sha, rsa_rc4_40_md5, rsa_rc2_40_md5, rsa_null_md5. Allowed TLS values are rsa_des_56_sha, rsa_rc4_56_sha`.
`tls-enabled`	`true`	(optional) Determines whether TLS is enabled.

steady-pool-size

Specifies the initial and minimum number of bean instances that are maintained in the pool. Default is 32. Applies to stateless session beans and message-driven beans.

Superelements

"bean-pool" on page 303 (`sun-ejb-jar.xml`)

Subelements

none - contains data

store-properties

Specifies session persistence (storage) properties.

Superelements

"session-manager" on page 390 (`sun-web.xml`)

Subelements

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

TABLE A–108 `store-properties` Subelements

Element	Required	Description
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.

Properties

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

TABLE A–109 `store-properties` Properties

Property	Default	Description
`directory`	*domain-dir*`/generated/jsp/j2ee-apps/`*app-name*`/`*app-name*`_war`	Specifies the absolute or relative pathname of the directory into which individual session files are written. A relative path is relative to the temporary work directory for this web application. Applicable only if the `persistence-type` attribute of the parent "session-manager" on page 390 element is `file`.

stub-property

Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Stub` object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC `Stub` implementation.

Superelements

"port-info" on page 366 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

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

TABLE A–110 `stub-property` subelements

Element	Required	Description
"name" on page 361	only one	Specifies the name of the entity.
"value" on page 405	only one	Specifies the value of the entity.

Example

```
<service-ref>
<service-ref-name>service/FooProxy</service-ref-name>
  <port-info>
       <service-endpoint-interface>a.FooPort</service-endpoint-interface>
       <wsdl-port>
          <namespaceURI>urn:Foo</namespaceURI>
          <localpart>FooPort</localpart>
       </wsdl-port>
       <stub-property>
          <name>javax.xml.rpc.service.endpoint.address</name>
          <value>http://localhost:8080/a/Foo</value>
       </stub-property>
  </port-info>
</service-ref>
```

sun-application

Defines the Application Server specific configuration for an application. This is the root element; there can only be one `sun-application` element in a `sun-application.xml` file. See "The sun-application.xml File" on page 285.

Superelements

none

Subelements

The following table describes subelements for the `sun-application` element.

TABLE A–111 `sun-application` Subelements

Element	Required	Description
"web" on page 407	zero or more	Specifies the application's web tier configuration.

TABLE A–111 `sun-application` Subelements *(Continued)*

Element	Required	Description
"pass-by-reference" on page 364	zero or one	Determines whether EJB modules use pass-by-value or pass-by-reference semantics.
"unique-id" on page 404	zero or one	Contains the unique ID for the application.
"security-role-mapping" on page 386	zero or more	Maps a role in the corresponding J2EE XML file to a user or group.
"realm" on page 373	zero or one	Specifies an authentication realm.

sun-application-client

Defines the Application Server specific configuration for an application client. This is the root element; there can only be one `sun-application-client` element in a `sun-application-client.xml` file. See "The sun-application-client.xml file" on page 297.

Superelements

none

Subelements

The following table describes subelements for the `sun-application-client` element.

TABLE A–112 `sun-application-client` subelements

Element	Required	Description
"ejb-ref" on page 331	zero or more	Maps the absolute JNDI name to the `ejb-ref` in the corresponding J2EE XML file.
"resource-ref" on page 379	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding J2EE XML file.
"resource-env-ref" on page 378	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding J2EE XML file.
"service-ref" on page 388	zero or more	Specifies runtime settings for a web service reference.
"message-destination" on page 355	zero or more	Specifies the name of a logical message destination.

sun-cmp-mapping

Specifies beans mapped to a particular database schema.

Note – A bean cannot be related to a bean that maps to a different database schema, even if the beans are deployed in the same EJB JAR file.

Superelements

"sun-cmp-mappings" on page 397 (`sun-cmp-mappings.xml`)

Subelements

The following table describes subelements for the `sun-cmp-mapping` element.

TABLE A–113 `sun-cmp-mapping` Subelements

Element	Required	Description
"schema" on page 383	only one	Specifies the file that contains a description of the database schema.
"entity-mapping" on page 334	one or more	Specifies the mapping of a bean to database columns.

sun-cmp-mappings

Defines the Application Server specific CMP mapping configuration for an EJB JAR file. This is the root element; there can only be one `sun-cmp-mappings` element in a `sun-cmp-mappings.xml` file. See "The sun-cmp-mappings.xml File" on page 293.

Superelements

none

Subelements

The following table describes subelements for the `sun-cmp-mappings` element.

TABLE A–114 `sun-cmp-mappings` Subelements

Element	Required	Description
"sun-cmp-mapping" on page 397	one or more	Specifies beans mapped to a particular database schema.

sun-ejb-jar

Defines the Application Server specific configuration for an EJB JAR file. This is the root element; there can only be one `sun-ejb-jar` element in a `sun-ejb-jar.xml` file. See "The sun-ejb-jar.xml File" on page 288.

Superelements

none

Subelements

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

TABLE A–115 `sun-ejb-jar` Subelements

Element	Required	Description
"security-role-mapping" on page 386	zero or more	Maps a role in the corresponding J2EE XML file to a user or group.
"enterprise-beans" on page 332	only one	Describes all the runtime properties for an EJB JAR file in the application.

sun-web-app

Defines Application Server specific configuration for a web module. This is the root element; there can only be one `sun-web-app` element in a `sun-web.xml` file. See "The sun-web.xml File" on page 285.

Superelements

none

Subelements

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

TABLE A–116 `sun-web-app` Subelements

Element	Required	Description
"context-root" on page 322	zero or one	Contains the web context root for the web application.

TABLE A–116 `sun-web-app` Subelements *(Continued)*

Element	Required	Description
"security-role-mapping" on page 386	zero or more	Maps roles to users or groups in the currently active realm.
"servlet" on page 389	zero or more	Specifies a principal name for a servlet, which is used for the `run-as` role defined in `web.xml`.
"idempotent-url-pattern" on page 339	zero or more	Enterprise Edition only. Do not use.
"session-config" on page 390	zero or one	Specifies session manager, session cookie, and other session-related information.
"ejb-ref" on page 331	zero or more	Maps the absolute JNDI name to the `ejb-ref` in the corresponding J2EE XML file.
"resource-ref" on page 379	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding J2EE XML file.
"resource-env-ref" on page 378	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding J2EE XML file.
"service-ref" on page 388	zero or more	Specifies runtime settings for a web service reference.
"cache" on page 304	zero or one	Configures caching for web application components.
"class-loader" on page 311	zero or one	Specifies class loader configuration information.
"jsp-config" on page 342	zero or one	Specifies JSP configuration information.
"locale-charset-info" on page 347	zero or one	Deprecated. Use the `parameter-encoding` subelement of `sun-web-app` instead.
"property (with attributes)" on page 368	zero or more	Specifies a property, which has a name and a value.
"parameter-encoding" on page 363	zero or one	Determines the default request character encoding and how the web container decodes parameters from forms according to a hidden field value.
"message-destination" on page 355	zero or more	Specifies the name of a logical message destination.
"webservice-description" on page 407	zero or more	Specifies a name and optional publish location for a web service.

Attributes

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

TABLE A–117 `sun-web-app` Attributes

Attribute	Default	Description
`error-url`	(blank)	(optional) Specifies a redirect URL in case of an error.

Properties

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

TABLE A–118 `sun-web-app` Properties

Property	Default	Description
`allowLinking`	`true`	If `true`, resources in this web application that are symbolic links are served.
`crossContextAllowed`	`true`	If `true`, allows this web application to access the contexts of other web applications using the `ServletContext.getContext()` method.
`relativeRedirectAllowed`	`false`	If `true`, allows this web application to send a relative URL to the client using `HttpServletResponse.sendRedirect()`, and instructs the web container not to translate any relative URLs to fully qualified ones.
`reuseSessionID`	`false`	If `true`, sessions generated for this web application use the session ID specified in the request.
`singleThreadedServletPoolSize`	`5`	Specifies the maximum number of servlet instances allocated for each `SingleThreadModel` servlet in the web application.

TABLE A–118 `sun-web-app` Properties *(Continued)*

Property	Default	Description
`tempdir`	*domain-dir*`/generated/j2ee-apps/`*app-name* or *domain-dir*`/generated/j2ee-modules/`*module-name*	Specifies a temporary directory for use by this web module. This value is used to construct the value of the `javax.servlet.context.tempdir` context attribute. Compiled JSP files are also placed in this directory.
`useResponseCTForHeaders`	`false`	If `true`, response headers are encoded using the response's charset instead of the default (UTF-8).

T

table-name

Specifies the name of a database table. The table must be present in the database schema file. See "Automatic Database Schema Capture" on page 176.

Superelements

"entity-mapping" on page 334, "secondary-table" on page 385 (`sun-cmp-mappings.xml`)

Subelements

none - contains data

target-server

Defines the IIOP listener configuration of the target server.

Superelements

"client-container" on page 313 (`sun-acc.xml`)

Subelements

The following table describes subelements for the `target-server` element.

TABLE A–119 `target-server` subelements

Element	Required	Description
"description" on page 326	zero or one	Specifies the description of the target server.
"security" on page 385	zero or one	Specifies the security configuration for the IIOP/SSL communication with the target server.

Attributes

The following table describes attributes for the `target-server` element.

TABLE A–120 `target-server` attributes

Attribute	Default	Description
`name`	none	Specifies the name of the application server instance accessed by the client container.
`address`	none	Specifies the host name or IP address (resolvable by DNS) of the server to which this client attaches.
`port`	none	Specifies the naming service port number of the server to which this client attaches. For a new server instance, assign a port number other than 3700. You can change the port number in the Administration Console. See the *Sun Java System Application Server Platform Edition 8.2 Administration Guide* for more information.

tie-class

Specifies the automatically generated name of a tie implementation class for a port component.

Superelements

"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

timeout

Specifies the "cache-mapping" on page 308 specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. If not specified, the default is the value of the `timeout` attribute of the "cache" on page 304 element.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none - contains data

Attributes

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

TABLE A–121 `timeout` Attributes

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

transport-config

Specifies the security transport information.

Superelements

"ior-security-config" on page 339 (`sun-ejb-jar.xml`)

Subelements

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

TABLE A–122 `transport-config` Subelements

Element	Required	Description
"integrity" on page 339	only one	Specifies if the target supports integrity-protected messages. The values are NONE, SUPPORTED, or REQUIRED.

TABLE A–122 `transport-config` Subelements *(Continued)*

Element	Required	Description
"confidentiality" on page 319	only one	Specifies if the target supports privacy-protected messages. The values are NONE, SUPPORTED, or REQUIRED.
"establish-trust-in-target" on page 335	only one	Specifies if the target is capable of authenticating *to* a client. The values are NONE, SUPPORTED, or REQUIRED.
"establish-trust-in-client" on page 334	only one	Specifies if the target is capable of authenticating a client. The values are NONE, SUPPORTED, or REQUIRED.

transport-guarantee

Specifies that the communication between client and server is `NONE`, `INTEGRAL`, or `CONFIDENTIAL`.

- `NONE` means the application does not require any transport guarantees.
- `INTEGRAL` means the application requires that the data sent between client and server be sent in such a way that it can't be changed in transit.
- `CONFIDENTIAL` means the application requires that the data be transmitted in a fashion that prevents other entities from observing the contents of the transmission.

In most cases, a value of `INTEGRAL` or `CONFIDENTIAL` indicates that the use of SSL is required.

Superelements

"webservice-endpoint" on page 408 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

U

unique-id

Contains the unique ID for the application. This value is automatically updated each time the application is deployed or redeployed. Do not edit this value.

Superelements

"sun-application" on page 395 (`sun-application.xml`), "enterprise-beans" on page 332 (`sun-ejb-jar.xml`)

Subelements

none - contains data

url-pattern

Specifies a servlet URL pattern for which caching is enabled. See the Servlet 2.4 specification section SRV. 11.2 for applicable patterns.

Superelements

"cache-mapping" on page 308 (`sun-web.xml`)

Subelements

none - contains data

use-thread-pool-id

Specifies the thread pool from which threads are selected for remote invocations of this bean.

Superelements

"ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

none - contains data

V

value

Specifies the value of the entity.

Superelements

"call-property" on page 309, "stub-property" on page 394 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`); "property (with subelements)" on page 369 (`sun-ejb-jar.xml`)

Subelements

none - contains data

victim-selection-policy

Specifies how stateful session beans are selected for passivation. Possible values are First In, First Out (`FIFO`), Least Recently Used (`LRU`), Not Recently Used (`NRU`). The default value is `NRU`, which is actually pseudo-LRU.

Note – You cannot plug in your own victim selection algorithm.

The victims are generally passivated into a backup store (typically a file system or database). This store is cleaned during startup, and also by a periodic background process that removes idle entries as specified by `removal-timeout-in-seconds`. The backup store is monitored by a background thread (or sweeper thread) to remove unwanted entries.

Applies to stateful session beans.

Superelements

"bean-cache" on page 302 (`sun-ejb-jar.xml`)

Subelements

none - contains data

Example

```
<victim-selection-policy>LRU</victim-selection-policy>
```

If both SSL2 and SSL3 are enabled, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption.

W

web

Specifies the application's web tier configuration.

Superelements

"sun-application" on page 395 (`sun-application.xml`)

Subelements

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

TABLE A–123 `web` Subelements

Element	Required	Description
"web-uri" on page 407	only one	Contains the web URI for the application.
"context-root" on page 322	only one	Contains the web context root for the application.

web-uri

Contains the web URI for the application. Must match the corresponding element in the `application.xml` file.

Superelements

"web" on page 407 (`sun-application.xml`)

Subelements

none - contains data

webservice-description

Specifies a name and optional publish location for a web service.

Superelements

"sun-web-app" on page 398 (`sun-web.xml`), "enterprise-beans" on page 332 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `webservice-description` element.

TABLE A–124 `webservice-description` subelements

Element	Required	Description
"webservice-description-name" on page 408	only one	Specifies a unique name for the web service within a web or EJB module.
"wsdl-publish-location" on page 410	zero or one	Specifies the URL of a directory to which a web service's WSDL is published during deployment.

webservice-description-name

Specifies a unique name for the web service within a web or EJB module.

Superelements

"webservice-description" on page 407 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

webservice-endpoint

Specifies information about a web service endpoint.

Superelements

"servlet" on page 389 (`sun-web.xml`), "ejb" on page 327 (`sun-ejb-jar.xml`)

Subelements

The following table describes subelements for the `webservice-endpoint` element.

TABLE A–125 `webservice-endpoint` subelements

Element	Required	Description
"port-component-name" on page 365	only one	Specifies a unique name for a port component within a web or EJB module.

TABLE A–125 `webservice-endpoint` subelements *(Continued)*

Element	Required	Description
"endpoint-address-uri" on page 331	zero or one	Specifies the automatically generated endpoint address.
"login-config" on page 351	zero or one	Specifies the authentication configuration for an EJB web service endpoint.
"message-security-binding" on page 357	zero or one	Specifies a custom authentication provider binding.
"transport-guarantee" on page 404	zero or one	Specifies that the communication between client and server is `NONE`, `INTEGRAL`, or `CONFIDENTIAL`.
"service-qname" on page 387	zero or one	Specifies the WSDL service element that is being referenced.
"tie-class" on page 402	zero or one	Specifies the automatically generated name of a tie implementation class for a port component.
"servlet-impl-class" on page 389	zero or one	Specifies the automatically generated name of the generated servlet implementation class.

wsdl-override

Specifies a valid URL pointing to a final WSDL document. If not specified, the WSDL document associated with the `service-ref` in the standard J2EE deployment descriptor is used.

Superelements

"service-ref" on page 388 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

none - contains data

Example

```
// available via HTTP
<wsdl-override>http://localhost:8000/myservice/myport?WSDL</wsdl-override>

// in a file
<wsdl-override>file:/home/user1/myfinalwsdl.wsdl</wsdl-override>
```

wsdl-port

Specifies the WSDL port.

Superelements

"port-info" on page 366 (`sun-web.xml`, `sun-ejb-jar.xml`, `sun-application-client.xml`)

Subelements

The following table describes subelements for the `wsdl-port` element.

TABLE A–126 `wsdl-port` subelements

Element	Required	Description
"namespaceURI" on page 361	only one	Specifies the namespace URI.
"localpart" on page 349	only one	Specifies the local part of a QNAME.

wsdl-publish-location

Specifies the URL of a directory to which a web service's WSDL is published during deployment. Any required files are published to this directory, preserving their location relative to the module-specific WSDL directory (`META-INF/wsdl` or `WEB-INF/wsdl`).

Superelements

"webservice-description" on page 407 (`sun-web.xml`, `sun-ejb-jar.xml`)

Subelements

none - contains data

Example

Suppose you have an `ejb.jar` file whose `webservices.xml` file's `wsdl-file` element contains the following reference:

```
META-INF/wsdl/a/Foo.wsdl
```

Suppose your `sun-ejb-jar` file contains the following element:

```
<wsdl-publish-location>file:/home/user1/publish</wsdl-publish-location>
```

The final WSDL is stored in `/home/user1/publish/a/Foo.wsdl`.

Index

B

C

D

E

F

G

K

L

M

N

O

P

Q

R

S

T

U

V

W

X