Sun GlassFish Enterprise Server v3 Embedded Server Guide

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

Part No: 821–1208–12
January 2010

100114@23031

Contents

Preface

This document explains how to use embedded Sun GlassFish™ Enterprise Server to run applications in embedded Enterprise Server and to develop applications in which Enterprise Server is embedded. This document is for software developers who are developing applications to run in embedded Enterprise Server. The ability to program in the Java ™ language is assumed.

This preface contains information about and conventions for the entire Sun GlassFish Enterprise Server (Enterprise Server) documentation set.

Enterprise Server v3 is developed through the GlassFish project open-source community at `https://glassfish.dev.java.net/`. The GlassFish project provides a structured process for developing the Enterprise Server platform that makes the new features of the Java EE platform available faster, while maintaining the most important feature of Java EE: compatibility. It enables Java developers to access the Enterprise Server source code and to contribute to the development of the Enterprise Server. The GlassFish project is designed to encourage communication between Sun engineers and the community.

The following topics are addressed here:

Enterprise Server Documentation Set

The Enterprise Server documentation set describes deployment planning and system installation. The Uniform Resource Locator (URL) for Enterprise Server documentation is `http://docs.sun.com/coll/1343.9`. For an introduction to Enterprise Server, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Enterprise Server Documentation Set

Book Title	Description
Release Notes	Provides late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of the supported hardware, operating system, Java™ Development Kit (JDK™), and database drivers.
Quick Start Guide	Explains how to get started with the Enterprise Server product.
Installation Guide	Explains how to install the software and its components.
Upgrade Guide	Explains how to upgrade to the latest version of Enterprise Server. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Administration Guide	Explains how to configure, monitor, and manage Enterprise Server subsystems and components from the command line by using the `asadmin(1M)` utility. Instructions for performing these tasks from the Administration Console are provided in the Administration Console online help.
Application Deployment Guide	Explains how to assemble and deploy applications to the Enterprise Server and provides information about deployment descriptors.
Your First Cup: An Introduction to the Java EE Platform	Provides a short tutorial for beginning Java EE programmers that explains the entire process for developing a simple enterprise application. The sample application is a web application that consists of a component that is based on the Enterprise JavaBeans™ specification, a JAX-RS web service, and a JavaServer™ Faces component for the web front end.
Application Development Guide	Explains how to create and implement Java Platform, Enterprise Edition (Java EE platform) applications that are intended to run on the Enterprise Server. These applications follow the open Java standards model for Java EE components and APIs. This guide provides information about developer tools, security, and debugging.
Add-On Component Development Guide	Explains how to use published interfaces of Enterprise Server to develop add-on components for Enterprise Server. This document explains how to perform *only* those tasks that ensure that the add-on component is suitable for Enterprise Server.

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

Book Title	Description
Embedded Server Guide	Explains how to run applications in embedded Enterprise Server and to develop applications in which Enterprise Server is embedded.
Scripting Framework Guide	Explains how to develop scripting applications in languages such as Ruby on Rails and Groovy on Grails for deployment to Enterprise Server.
Troubleshooting Guide	Describes common problems that you might encounter when using Enterprise Server and how to solve them.
Error Message Reference	Describes error messages that you might encounter when using Enterprise Server.
Reference Manual	Provides reference information in man page format for Enterprise Server administration commands, utility commands, and related concepts.
Domain File Format Reference	Describes the format of the Enterprise Server configuration file, `domain.xml`.
Java EE 6 Tutorial, Volume I	Explains how to use Java EE 6 platform technologies and APIs to develop Java EE applications.
Message Queue Release Notes	Describes new features, compatibility issues, and existing bugs for Sun GlassFish Message Queue.
Message Queue Administration Guide	Explains how to set up and manage a Sun GlassFish Message Queue messaging system.
Message Queue Developer's Guide for JMX Clients	Describes the application programming interface in Sun GlassFish Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX).
System Virtualization Support in Sun Java System Products	Summarizes Sun support for Sun Java System products when used in conjunction with system virtualization products and features.

Related Documentation

The Java EE 6 Tutorial, Volume II (`https://www.sun.com/offers/details/java_ee6_tutorial.xml`) contains all the topics in *Java EE 6 Tutorial, Volume I* and adds advanced topics, additional technologies, and case studies. The document is available to registered users of Enterprise Server.

Javadoc™ tool reference documentation for packages that are provided with Enterprise Server is available as follows:

- The API specification for version 6 of Java EE is located at `http://java.sun.com/javaee/6/docs/api/`.
- The API specification for Enterprise Server v3, including Java EE 6 platform packages and nonplatform packages that are specific to the Enterprise Server product, is located at: `https://glassfish.dev.java.net/nonav/docs/v3/api/`.

Additionally, the following resources might be useful:

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

For information about creating enterprise applications in the NetBeans™ Integrated Development Environment (IDE), see `http://www.netbeans.org/kb/60/index.html`.

For information about the Java DB for use with the Enterprise Server, see `http://developers.sun.com/javadb/`.

The sample applications demonstrate a broad range of Java EE technologies. The samples are bundled with the Java EE Software Development Kit (SDK).

Typographic Conventions

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

TABLE P–2 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–3 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.

Default Paths and File Names

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

TABLE P–4 Default Paths and File Names

Placeholder	Description	Default Value
as-install	Represents the base installation directory for Enterprise Server. In configuration files, *as-install* is represented as follows: `${com.sun.aas.installRoot}`	Installations on the Solaris™ operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfishv3/glassfish` Windows, all installations: *SystemDrive*`:\glassfishv3\glassfish`
as-install-parent	Represents the parent of the base installation directory for Enterprise Server.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfishv3` Windows, all installations: *SystemDrive*`:\glassfishv3`
domain-root-dir	Represents the directory in which a domain is created by default.	*as-install*`/domains/`

TABLE P–4 Default Paths and File Names *(Continued)*

Placeholder	Description	Default Value
domain-dir	Represents the directory in which a domain's configuration is stored. In configuration files, *domain-dir* is represented as follows: `${com.sun.aas.instanceRoot}`	*domain-root-dir*/*domain-name*

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 821-1208.

Sun GlassFish™ Enterprise Server v3 Embedded Server Guide

This document explains how to use embedded Sun GlassFish Enterprise Server to run applications in embedded Enterprise Server and to develop applications in which Enterprise Server is embedded. This document is for software developers who are developing applications to run in embedded Enterprise Server. The ability to program in the Java ™ language is assumed.

This document addresses the following topics:

Introduction to Embedded Enterprise Server

Embedded Sun GlassFish Enterprise Server enables you to use Enterprise Server as a library. Embedded Enterprise Server also enables you to run Enterprise Server inside any Virtual Machine for the Java™ platform (Java Virtual Machine or JVM™machine).

No installation or configuration of embedded Enterprise Server is required. The ability to run Enterprise Server inside applications without installation or configuration simplifies the process of bundling Enterprise Server with applications.

Note – Embedded Enterprise Server does *not* run on the Java Platform, Micro Edition (Java ME platform).

You can use embedded Enterprise Server in the following ways:

- With the Sun GlassFish Embedded Server API
- With the Maven Plug-in
- With the EJB 3.1 Embeddable API

Embedded Enterprise Server provides a plug-in for Apache Maven. This plug-in simplifies the testing of applications by enabling you to build an application and run it with Enterprise Server on a single system. Testing and debugging are simplified because the need for an installed and configured Enterprise Server is eliminated. Therefore, you can run unit tests automatically in every build.

Note – Using Embedded Enterprise Server does not involve any use of OSGi technology.

Including the Sun GlassFish Embedded Server API in Applications

Sun GlassFish Enterprise Server provides an application programming interface (API) for developing applications in which Enterprise Server is embedded. For details, see the `org.glassfish.api.embedded.*` packages at `https://glassfish.dev.java.net/nonav/docs/v3/api/`.

Developing an application in which Enterprise Server is embedded is explained in the following topics:

Setting the Class Path

To enable your applications to locate the class libraries for embedded Enterprise Server, add one of the following JAR files to your class path:

`glassfish-embedded-web.jar`
: Contains classes needed for deploying Java EE web applications. Download this file from `http://download.java.net/maven/glassfish/org/glassfish/extras/`.

`glassfish-embedded-all.jar`
: Contains classes needed for deploying all Java EE application types. Download this file from `http://download.java.net/maven/glassfish/org/glassfish/extras/`.

`glassfish-embedded-static-shell.jar`
Contains references to classes needed for deploying all Java EE application types. Must be used with a nonembedded installation of Enterprise Server. Obtain this file from the *as-install*`glassfish/lib/embedded` directory of a nonembedded Enterprise Server installation. For an explanation of *as-install*, see Installation Root Directory.

In addition, add to the class path any other JAR files or classes upon which your applications depend. For example, if an application uses a database other than Java DB, include the Java DataBase Connectivity (JDBC™) driver JAR files in the class path.

Creating, Starting, and Stopping Embedded Enterprise Server

Before you can run applications, you must set up and run the embedded Enterprise Server. This section includes the following topics:

Creating and Configuring an Embedded Enterprise Server

To create configure an embedded Enterprise Server, perform these tasks:

1. Instantiate the `org.glassfish.api.embedded.Server.Builder` class.
2. Invoke any methods for configuration settings that you require. This is optional.
3. Invoke one of the `build` methods to create a `Server` object.

The methods of this class for setting the server configuration are listed in the following table. The default value of each configuration setting is also listed.

TABLE 1 Constructor and Methods of the `Server.Builder` Class

Purpose	Method	Default Value
Creates a server builder and names the server	`Server.Builder(java.lang.String` *id*`)`	None
References an embedded file system	`embeddedFileSystem(EmbeddedFileSystem` *file-system*`)`	None
Enables verbose mode	`verbose(boolean` *enabled*`)`	`true`
Enables logging	`logger(boolean` *enabled*`)`	`true`

TABLE 1 Constructor and Methods of the `Server.Builder` Class *(Continued)*

Purpose	Method	Default Value
Specifies a log file	`logFile(File` *log-file*`)`	*instance-dir*`logs/server.log` (see Instance Root Directory)
Creates a server	`build()`	None
Creates a server with properties	`build(Properties` *properties*`)`	None

EXAMPLE 1 Creating an Embedded Enterprise Server

This example shows code for creating a server and enabling logging.

```
...
import org.glassfish.api.embedded.*;
...
    Server.Builder builder = new Server.Builder("test");
    builder.logger(true);
    ...
    Server server = builder.build();
...
```

Specifying an Embedded Enterprise Server File System

Specifying an embedded file system for an embedded Enterprise Server is optional. What you don't specify is created automatically and used transparently by the server. An embedded file system enables you to do the following:

- To use the configuration information of an existing installation of Enterprise Server
- To control where embedded Enterprise Server creates its configuration directories
- To specify a configuration file, which controls how embedded Enterprise Server functions

To specify an embedded file system for embedded Enterprise Server, perform these tasks:

1. Instantiate the `org.glassfish.api.embedded.EmbeddedFileSystem.Builder` class.
2. Invoke any methods for configuration settings that you require. This is optional.
3. Invoke the `build` method to create an `EmbeddedFileSystem` object.

If you are including the `glassfish-embedded-static-shell.jar` file in your class path, the methods of the `EmbeddedFileSystem.Builder` class can only point to the referenced installation. These methods cannot create or delete any directories or files.

The methods of the `EmbeddedFileSystem.Builder` class for setting the embedded file system configuration are listed in the following table. The default value of each configuration setting is also listed.

TABLE 2 Constructor and Methods of the `EmbeddedFileSystem.Builder` Class

Purpose	Method	Default Value
Creates an embedded file system builder	`EmbeddedFileSystem.Builder()`	None
Deletes embedded file system files when the server is stopped **Caution –** *Do not* set `autoDelete` to `true` if you are using `installRoot` to refer to a preexisting Enterprise Server installation.	`autoDelete(boolean` *enabled*`)`	`false`
Creates a new or references an existing Installation Root Directory	`installRoot(java.io.File` *install-root*`)`	In order of precedence: ■ `glassfish.embedded.tmpdir` system property value ■ `user.dir` system property value ■ Current directory
Creates or references an Installation Root Directory in which the embedded server and file system use different class loaders if *cooked-mode* is false	`installRoot(java.io.File` *install-root*`, boolean` *cooked-mode*`)`	Same as the other `installRoot` method, *cooked-mode* is `true`
Creates a new or references an existing Instance Root Directory	`instanceRoot(java.io.File` *instance-root*`)`	*as-install*`domains/domain1`
Creates a new or references an existing configuration file	`configurationFile(java.io.File` *config-file*`)`	*instance-dir*`config/domain.xml`
Creates or references a configuration file that is read-only if *read-only* is true	`configurationFile(java.io.File` *config-file* `boolean` *read-only*`)`	*instance-dir*`config/domain.xml`, *read-only* is `false`
Creates an embedded file system	`build()`	None

EXAMPLE 2 Creating an Embedded File System

This example shows code for creating an embedded file system whose configuration information is stored in the file `C:\samples\test\applicationserver\domains\domain1\config\domain.xml`. This example also includes the code from Example 1 for creating `Server.Builder` and `Server` objects.

```
...
import java.io.File;
...
import org.glassfish.api.embedded.*;
...
    File installDir = new File("c:\\samples\\testapp\\applicationserver");
    File domainDir = new File(installDir, "domains\\domain1");
        ...
    File domainConfig = new File(domainDir, "config");
    File domainXml = new File(domainConfig, "domain.xml");
    ...
    Server.Builder builder = new Server.Builder("test");
    ...
    EmbeddedFileSystem.Builder efsb = new EmbeddedFileSystem.Builder();
    efsb.installRoot(installDir);
    efsb.instanceRoot(domainDir);
    efsb.configurationFile(domainXml);
    EmbeddedFileSystem efs = efsb.build();
    builder.embeddedFileSystem(efs);
    ...
    Server server = builder.build();
...
```

Installation Root Directory

The installation root directory, represented as *as-install*, is the parent of the directory that embedded Sun GlassFish Enterprise Server uses for configuration files. This directory corresponds to the base directory for an installation of Enterprise Server. Configuration files are contained in the following directories in the base directory for an installation of Enterprise Server:

- `domains`
- `lib`

Specify the installation root directory if any of the following conditions applies:

- You require Sun GlassFish Enterprise Server to create the directory for configuration files at a specific location.

- You are using an existing domain that is at the default location, that is contained in the `domains` subdirectory of the installation root directory. The `domains` subdirectory must contain only one domain directory. This domain can be part of a nonembedded installation of Sun GlassFish Enterprise Server.

  If the `domains` subdirectory contains multiple directories, or if you are using a domain at a non-default location, you must also specify the instance root directory.

Note – If the instance root directory is also specified, configuration files for the domain are obtained from the instance root directory, not the `domains` directory under the installation root directory.

How embedded Sun GlassFish Enterprise Server uses the installation root directory depends on the content of this directory:

- If the installation root directory contains domain configuration files, embedded Sun GlassFish Enterprise Server reads the configuration files.
- If the installation root directory does not contain domain configuration files, embedded Sun GlassFish Enterprise Server copies configuration files into this directory.
- If the installation root directory does not exist, embedded Sun GlassFish Enterprise Server creates the directory and copies configuration files into the directory.

If the installation root directory is not specified, embedded Sun GlassFish Enterprise Server creates a directory named `gfembed`*random-number*`tmp` in the current working directory, where *random-number* is a randomly generated 19-digit number. Embedded Sun GlassFish Enterprise Server then copies configuration files into this directory.

Instance Root Directory

The instance root directory, represented as *instance-dir*, is the parent directory of a server instance directory. Embedded Sun GlassFish Enterprise Server uses the server instance directory for domain configuration files.

Specify the instance root directory if any of the following conditions applies:

- You are using a domain directory that is at a non-default location, that is not contained in the `domains` subdirectory of the installation root directory.

  For example, if your domain directory is at `/tmp/domain1`, specify the instance root directory as `/tmp/domain1`.
- The `domains` subdirectory of your installation root directory contains multiple domain directories.

For example, the `domains` subdirectory of the `/home/gfuser/glassfish` installation root directory might contain the domain directories `domain1` and `domain2`. To use the `domain` directory `domain2`, specify the instance root directory as `/home/gfuser/glassfish/domains/domain2`.

How embedded Sun GlassFish Enterprise Server uses the instance root directory depends on the content of this directory:

- If the instance root directory contains domain configuration files, embedded Sun GlassFish Enterprise Server reads the configuration files.
- If the instance root directory does not contain domain configuration files, embedded Sun GlassFish Enterprise Server copies configuration files into this directory.
- If the instance root directory does not exist, embedded Sun GlassFish Enterprise Server creates the directory and copies configuration files into the directory.

If the instance root directory is not specified, embedded Sun GlassFish Enterprise Server uses the `domains` subdirectory of the installation root directory for domain configuration files.

Using an Existing `domain.xml` File

Using an existing `domain.xml` file avoids the need to configure embedded Enterprise Server programmatically in your application. Your application obtains domain configuration data from an existing `domain.xml` file. You can create this file by using the administrative interfaces of an installation of nonembedded Enterprise Server. To specify an existing `domain.xml` file, invoke the `installRoot`, `instanceRoot`, or `configurationFile` method of the `EmbeddedFileSystem.Builder` class or a combination of these methods.

Running an Embedded Enterprise Server

After you create an embedded Enterprise Server as described in "Creating and Configuring an Embedded Enterprise Server" on page 15, you can perform operations such as:

- Setting the Port
- Starting the server
- Stopping the server

Setting the Port of an Embedded Enterprise Server From an Application

You must set the server's HTTP port. If you do not set the port, your application fails to start and throws an exception. You can set the port directly or indirectly.

- To set the port directly, invoke the `createPort` method of the `Server` object.
- To port indirectly, set up an embedded file system, which includes a `domain.xml` file that sets the port. For more information, see "Specifying an Embedded Enterprise Server File System" on page 16 and "Using an Existing `domain.xml` File" on page 20.

EXAMPLE 3 Starting an Embedded Enterprise Server

This example shows code for setting the port of an embedded Enterprise Server. This example also includes the code from Example 1 for creating `Server.Builder` and `Server` objects.

```
...
import org.glassfish.api.embedded.*;
...
    Server.Builder builder = new Server.Builder("test");
    ...
    Server server = builder.build();
    server.createPort(8080);
...
```

Starting an Embedded Enterprise Server From an Application

To start an embedded Enterprise Server, invoke the `start` method of the `Server` object.

EXAMPLE 4 Starting an Embedded Enterprise Server

This example shows code for setting the port and starting an embedded Enterprise Server. This example also includes the code from Example 1 for creating `Server.Builder` and `Server` objects.

```
...
import org.glassfish.api.embedded.*;
...
    Server.Builder builder = new Server.Builder("test");
    ...
    Server server = builder.build();
    server.createPort(8080);
    server.start();
...
```

Stopping an Embedded Enterprise Server From an Application

The API for embedded Enterprise Server provides a method for stopping an embedded server. Using this method enables your application to stop the server in an orderly fashion by performing any necessary cleanup steps before stopping the server, for example:

- Undeploying deployed applications
- Releasing any resources that your application uses

To stop an embedded Enterprise Server, invoke the `stop` method of an existing `Server` object.

EXAMPLE 5 Stopping an Embedded Enterprise Server

This example shows code for prompting the user to press the Enter key to stop an embedded Enterprise Server. When a user presses Enter, the application undeploys any deployed applications before stopping the server. For more information about undeploying applications, see "Undeploying an Application" on page 24. Code for creating a `Server` object is not shown in this example. For an example of code for creating a `Server` object, see Example 1.

```
...
import java.io.BufferedReader;
...
import org.glassfish.api.embedded.*;
...
    EmbeddedDeployer deployer = server.getDeployer();
    ...
    System.out.println("Press Enter to stop server");
        // wait for Enter
    new BufferedReader(new java.io.InputStreamReader(System.in)).readLine();
    deployer.undeployAll();
    server.stop();
...
```

Deploying and Undeploying an Application in an Embedded Enterprise Server

Deploying an application installs the files that comprise the application into Enterprise Server and makes the application ready to run. By default, an application is enabled when it is deployed.

For general information about deploying applications in Enterprise Server, see *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.

▼ To Deploy an Application From an Archive File or a Directory

An archive file contains the resources, deployment descriptor, and classes of an application. The content of the file must be organized in the directory structure that the Java EE specifications define for the type of archive that the file contains. For more information, see Chapter 2, "Deploying Applications," in *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.

Deploying an application from a directory enables you to deploy an application without the need to package the application in an archive file. The contents of the directory must match the contents of the expanded Java EE archive file as laid out by the Enterprise Server. The directory must be accessible to the machine on which the *deploying* application runs. For more information about the requirements for deploying an application from a directory, see "To Deploy an Application or Module in a Directory Format" in *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.

1. **Instantiate the `java.io.File` class to represent the archive file or directory.**

2. **Invoke the `getDeployer` method of the `Server` object to get an instance of the `org.glassfish.api.embedded.EmbeddedDeployer` class.**

3. **Instantiate a `org.glassfish.api.deployment.DeployCommandParameters` class.**

   To use the default parameter settings, instantiate an empty `DeployCommandParameters` class. For information about the fields in this class that you can set, see the descriptions of the equivalent `deploy(1)` command parameters.

4. **Invoke the `deploy(File` *archive*`, DeployCommandParameters` *params*`)` method of the instance of the `EmbeddedDeployer` object.**

   Specify the `java.io.File` and `DeployCommandParameters` class instances you created previously as the method parameters.

Example 6 Deploying an Application From an Archive File

This example shows code for deploying an application from the archive file `c:\samples\simple.war` and setting the *contextroot* parameter of the `DeployCommandParameters` class. This example also includes the code from Example 1 for creating `Server.Builder` and `Server` objects.

```
...
import java.io.File;
...
import org.glassfish.api.deployment.*;
...
import org.glassfish.api.embedded.*;
...
    Server.Builder builder = new Server.Builder("test");
    ...
    Server server = builder.build();
    server.createPort(8080);
    server.start();

    File war = new File("c:\\samples\\simple.war");
    EmbeddedDeployer deployer = server.getDeployer();
    DeployCommandParameters params = new DeployCommandParameters();
    params.contextroot = "simple";
    deployer.deploy(war, params);
...
```

Undeploying an Application

Undeploy an application when the application is no longer required to run in Enterprise Server. For example, before stopping Enterprise Server, undeploy all applications that are running in Enterprise Server.

To undeploy an application, invoke the `undeploy` method of an existing `EmbeddedDeployer` object. In the method invocation, pass the name of the application and the name of its `DeployCommandParameters` class as parameters. Both are specified when the application is deployed.

To undeploy all deployed applications, invoke the `undeployAll` method of an existing `EmbeddedDeployer` object. This method takes no parameters.

EXAMPLE 7 Undeploying an Application

This example shows code for undeploying the application that was deployed in Example 6.

```
...
import org.glassfish.api.deployment.*;
...
import org.glassfish.api.embedded.*;
...
    deployer.undeploy(war, params);
...
```

Running `asadmin` Commands Using the Sun GlassFish Embedded Server API

Running `asadmin(1M)` commands from an application enables the application to configure the embedded Enterprise Server to suit the application's requirements. For example, an application can run the required `asadmin` commands to create a JDBC technology connection to a database.

For more information about configuring embedded Enterprise Server, see the *Sun GlassFish Enterprise Server v3 Administration Guide*. For detailed information about `asadmin` commands, see Section 1 of the *Sun GlassFish Enterprise Server v3 Reference Manual*.

Note – Ensure that your application has started an embedded Enterprise Server before the application attempts to run `asadmin` commands. For more information, see "Running an Embedded Enterprise Server" on page 20.

The `org.glassfish.api.admin` package contains classes that you can use to run `asadmin` commands. Use the following code examples as templates and change the command name, parameter names, and parameter values as needed.

EXAMPLE 8 Running an `asadmin create-jdbc-resource` Command

This example shows code for running an `asadmin create-jdbc-resource` command. Code for creating and starting the server is not shown in this example. For an example of code for creating and starting the server, see Example 4.

```
...
import org.glassfish.api.embedded.*;
import org.glassfish.api.admin.*;
...
    String command = "create-jdbc-resource";
    ParameterMap params = new ParameterMap();
    params.add("connectionpoolid", "DerbyPool");
    params.add("jndi_name", "jdbc/DerbyPool");
    CommandRunner runner = server.getHabitat().getComponent(CommandRunner.class);
    ActionReport report = server.getHabitat().getComponent(ActionReport.class);
    runner.getCommandInvocation(command, report).parameters(params).execute();
...
```

EXAMPLE 9 Running an `asadmin set-log-level` Command

This example shows code for running an `asadmin set-log-level` command. Code for creating and starting the server is not shown in this example. For an example of code for creating and starting the server, see Example 4.

```
...
import org.glassfish.api.embedded.*;
import org.glassfish.api.admin.*;
...
    String command = "set-log-level";
    ParameterMap params = new ParameterMap();
    params.add("javax\.enterprise\.system\.container\.web", "FINE");
    CommandRunner runner = server.getHabitat().getComponent(CommandRunner.class);
    ActionReport report = server.getHabitat().getComponent(ActionReport.class);
    runner.getCommandInvocation(command, report).parameters(params).execute();
...
```

For another way to change log levels, see "Changing Log Levels in Embedded Enterprise Server" on page 37.

Sample Applications

EXAMPLE 10 Using an Existing `domain.xml` File and Deploying an Application From an Archive File

This example shows code for the following:

EXAMPLE 10 Using an Existing `domain.xml` File and Deploying an Application From an Archive File *(Continued)*

- Using the existing file `c:\myapp\embeddedserver\domains\domain1\config\domain.xml` and preserving this file when the application is stopped.
- Deploying an application from the archive file `c:\samples\simple.war`.

The files that this example uses are organized as follows:

- `c:\myapp\embeddedserver\lib\glassfish-embedded-all.jar`
- `c:\myapp\embeddedserver\domains\domain1\config\domain.xml`
- `c:\myapp\embeddedserver\domains\domain1\logs`
- `c:\myapp\embeddedserver\domains\domain1\docroot`

```
import java.io.File;
import java.io.BufferedReader;
import org.glassfish.api.deployment.*;
import org.glassfish.api.embedded.*;

public class Main {

     /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {
        File installDir = new File ("c:\\myapp\\embeddedserver");
        File war = new File("c:\\samples\\simple.war");
        try {
            Server.Builder builder = new Server.Builder("test");
            ...
            EmbeddedFileSystem.Builder efsb = new EmbeddedFileSystem.Builder();
            efsb.autoDelete(false);
            efsb.installRoot(installDir);
            EmbeddedFileSystem efs = efsb.build();
            builder.embeddedFileSystem(efs);
            ...
            Server server = builder.build();
            server.createPort(8080);
            server.start();

            EmbeddedDeployer deployer = server.getDeployer();
            DeployCommandParameters params = new DeployCommandParameters();
            deployer.deploy(war, params);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
```

EXAMPLE 10 Using an Existing `domain.xml` File and Deploying an Application From an Archive File *(Continued)*

```
        System.out.println("Press Enter to stop server");
        // wait for Enter
        new BufferedReader(new java.io.InputStreamReader(System.in)).readLine();
        try {
            deployer.undeployAll();
            server.stop();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

▼ To Run Embedded Enterprise Server from the Command Line

After you have written a class that uses the Sun GlassFish Embedded Server API to start the server, deploy your applications, and stop the server, you can run this class at the command line.

Before You Begin Ensure that the following prerequisites are met:

- A distribution of embedded Enterprise Server is downloaded.
- A user-created class file that uses the Embedded Server API to start the server, deploy your applications, and stop the server is written.

● **Run embedded Enterprise Server in the Java application launcher, specifying the applications to deploy.**

`java -jar` *glassfish-embedded-jar embedded-class*

glassfish-embedded-jar
The full path to the file that contains your distribution of embedded Enterprise Server: `glassfish-embedded-web.jar`, `glassfish-embedded-all.jar`, or `glassfish-embedded-static-shell.jar`.

embedded-class
A user-created class file with a `main` method that uses the Embedded Server API to start the server, deploy your applications, and stop the server.

The applications continue to run in embedded Enterprise Server until embedded Enterprise Server is stopped.

Example 11 Running Embedded Enterprise Server

This example shows the command for running embedded Enterprise Server as follows:

- The embedded Enterprise Server JAR file is `glassfish-embedded-all.jar`.
- The user class file is `myembed.class`.

```
java -jar glassfish-embedded-all.jar myembed.class
```

Testing Applications with the Maven Plug-in for Embedded Enterprise Server

If you are using Apache Maven, the plug-in for embedded Enterprise Server simplifies the testing of applications. This plug-in enables you to build and start an unpackaged application with a single Maven goal.

Testing applications with the Maven plug-in involves the following tasks:

- "To Set Up Your Maven Environment" on page 28
- "To Build and Start an Application From Maven" on page 31
- "To Stop Embedded Enterprise Server" on page 32
- "To Redeploy an Application That Was Built and Started From Maven" on page 32

Predefined Maven goals for embedded Enterprise Server are described in "Maven Goals for Embedded Enterprise Server" on page 32.

▼ To Set Up Your Maven Environment

Setting up your Maven environment enables Maven to download the required embedded Enterprise Server distribution file when you build your project. Setting up your Maven environment also identifies the plug-in that enables you to build and start an unpackaged application with a single Maven goal.

Before You Begin Ensure that Apache Maven is installed.

1 Identify the Maven plug-in for embedded Enterprise Server.

Add the following `plugin` element to your POM file:

```
...
        <plugins>
            ...
            <plugin>
                <groupId>org.glassfish</groupId>
                <version>version</version>
```

```
                ...
            </plugin>
            ...
        </plugins>
...
```

version The version to use. The version of the final promoted build for this release is `3.0-74b`.

2 Configure the `embedded-glassfish` goal prefix, the application name, and other standard settings.

Add the following `configuration` element to your POM file:

```
...
        <plugins>
            ...
            <plugin>
                ...
                <configuration>
                    <goalPrefix>embedded-glassfish</goalPrefix>
                    ...
                    <app>test.war</app>
                    <port>8080</port>
                    <contextRoot>test</contextRoot>
                    <autoDelete>true</autoDelete>
                    ...
                </configuration>
                ...
            </plugin>
            ...
        </plugins>
...
```

In the *app* parameter, substitute the archive file or directory for your application. The optional *port*, *contextRoot*, and *autoDelete* parameters show example values. For details, see "Maven Goals for Embedded Enterprise Server" on page 32.

3 Configure Maven goals.

Add `execution` elements to your POM file:

```
...
        <plugins>
            ...
            <plugin>
                ...
                <executions>
                    <execution>
                        <phase>install</phase>
```

```
                        <goals>
                                <goal>goal</goal>
                        </goals>
                    </execution>
                </executions>
                ...
            </plugin>
            ...
        </plugins>
...
```

goal The goal to use. See "Maven Goals for Embedded Enterprise Server" on page 32.

4 Configure the repository.

Add the following `repository` element to your POM file:

```
<pluginRepositories>
    <pluginRepository>
        <id>maven2-repository.dev.java.net</id>
        <name>Java.net Repository for Maven</name>
        <url>http://download.java.net/maven/glassfish/</url>
    </pluginRepository>
</pluginRepositories>
```

Example 12 POM File for Configuring Maven to Use Embedded Enterprise Server

This example shows a POM file for configuring Maven to use embedded Enterprise Server.

```
<?xml version="1.0" encoding="UTF-8"?>
    Line breaks in the following element are for readability purposes only
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>org.glassfish</groupId>
  <artifactId>maven-glassfish-plugin-tester</artifactId>
  <version>3.0-74b</version>
  <name>Maven test</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.glassfish</groupId>
        <artifactId>maven-embedded-glassfish-plugin</artifactId>
        <version>3.0-74b</version>
        <configuration>
```

```
            <goalPrefix>embedded-glassfish</goalPrefix>
            <app>test.war</app>
            <port>8080</port>
            <contextRoot>test</contextRoot>
            <autoDelete>true</autoDelete>
          </configuration>
          <executions>
            <execution>
              <phase>install</phase>
              <goals>
                    <goal>run</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
    <pluginRepositories>
        <pluginRepository>
            <id>maven2-repository.dev.java.net</id>
            <name>Java.net Repository for Maven</name>
            <url>http://download.java.net/maven/glassfish/</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

▼ To Build and Start an Application From Maven

If you are using Maven to manage the development of your application, you can use a Maven goal to build and start the application in embedded Enterprise Server.

Before You Begin Ensure that your Maven environment is configured.

1. **Include the path to the Maven executable file `mvn` in your path statement.**
2. **Ensure that the `JAVA_HOME` environment variable is defined.**
3. **Create a directory for the Maven project for your application.**
4. **Copy to your project directory the POM file that you created in "To Set Up Your Maven Environment" on page 28.**
5. **Run the following command in your project directory:**

   ```
   mvn install
   ```

This command performs the following actions:

- Installs the Maven repository in a directory named `.m2` under your home directory.
- Starts embedded Enterprise Server.
- Deploys your application.

The application continues to run in embedded Enterprise Server until embedded Enterprise Server is stopped.

▼ To Stop Embedded Enterprise Server

1. **Change to the root directory of the Maven project for your application.**
2. **Run the Maven goal to stop the application in embedded Enterprise Server.**

   ```
   mvn embedded-glassfish:stop
   ```

   This runs the `stop` method of the `Server` object and any other methods that are required to shut down the server in an orderly fashion. See "Stopping an Embedded Enterprise Server From an Application" on page 21.

▼ To Redeploy an Application That Was Built and Started From Maven

An application that was built and started from Maven continues to run in embedded Enterprise Server until embedded Enterprise Server is stopped. While the application is running, you can test changes to the application by redeploying it.

- **In the window from where the application was built and started from Maven, press Enter.**

Maven Goals for Embedded Enterprise Server

You can use the following Maven goals to test your applications with embedded Enterprise Server:

`embedded-glassfish:run` Goal

This goal starts the server and deploys an application. You can redeploy if you change the application. The application can be a packaged archive or a directory that contains an exploded application. You can set the parameters described in the following table.

TABLE 3 `embedded-glassfish:run` Parameters

Parameter	Default	Description
serverID	`maven`	(optional) The ID of the server to start.
containerType	`all`	(optional) The container to start: `web`, `ejb`, `jpa`, or `all`.
installRoot	In order of precedence: ■ `glassfish.embedded.tmpdir` system property value ■ `user.dir` system property value ■ Current directory	(optional) The Installation Root Directory.
instanceRoot	*as-install*`domains/domain1`	(optional) The Instance Root Directory
configFile	*instance-dir*`config/domain.xml`	(optional) The configuration file.
port	None. Must be set explicitly or defined in the configuration file.	(optional) The HTTP port.
app	None.	The archive file or directory for the application to be deployed.
name	In order of precedence: ■ The `application-name` or `module-name` in the deployment descriptor. ■ The name of the archive file without the extension or the directory name. For more information, see "Naming Standards" in *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.	(optional) The name of the application.
contextRoot	The name of the application.	(optional) The context root of the application.
precompileJsp	`false`	(optional) If `true`, JSP pages are precompiled during deployment.
createTables	Value of the `create-tables-at-deploy` attribute in `sun-ejb-jar.xml`.	(optional) If `true`, creates database tables during deployment for beans that are automatically mapped by the EJB™ container.

TABLE 3 `embedded-glassfish:run` Parameters *(Continued)*

Parameter	Default	Description
autoDelete	`false`	(optional) If `true`, deletes the contents of the Instance Root Directory when the server is stopped. **Caution –** *Do not* set `autoDelete` to `true` if you are using `installRoot` to refer to a preexisting Enterprise Server installation.

`embedded-glassfish:start` Goal

This goal starts the server. You can set the parameters described in the following table.

TABLE 4 `embedded-glassfish:start` Parameters

Parameter	Default	Description
serverID	`maven`	(optional) The ID of the server to start.
containerType	`all`	(optional) The container to start: `web`, `ejb`, `jpa`, or `all`.
installRoot	In order of precedence: ■ `glassfish.embedded.tmpdir` system property value ■ `user.dir` system property value ■ Current directory	(optional) The Installation Root Directory.
instanceRoot	*as-install*`domains/domain1`	(optional) The Instance Root Directory
configFile	*instance-dir*`config/domain.xml`	(optional) The configuration file.
port	None. Must be set explicitly or defined in the configuration file.	(optional) The HTTP port.
autoDelete	`false`	(optional) If `true`, deletes the contents of the Instance Root Directory when the server is stopped. **Caution –** *Do not* set `autoDelete` to `true` if you are using `installRoot` to refer to a preexisting Enterprise Server installation.

`embedded-glassfish:deploy` Goal

This goal deploys an application. You can redeploy if you change the application. The application can be a packaged archive or a directory that contains an exploded application. You can set the parameters described in the following table.

TABLE 5 `embedded-glassfish:deploy` Parameters

Parameter	Default	Description
app	None.	The archive file or directory for the application to be deployed.
name	In order of precedence: ■ The `application-name` or `module-name` in the deployment descriptor. ■ The name of the archive file without the extension or the directory name. For more information, see "Naming Standards" in *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.	(optional) The name of the application.
contextRoot	The name of the application.	(optional) The context root of the application.
precompileJsp	`false`	(optional) If `true`, JSP pages are precompiled during deployment.
createTables	Value of the `create-tables-at-deploy` attribute in `sun-ejb-jar.xml`.	(optional) If `true`, creates database tables during deployment for beans that are automatically mapped by the EJB container.

`embedded-glassfish:undeploy` Goal

This goal undeploys an application. You can set the parameters described in the following table.

TABLE 6 `embedded-glassfish:undeploy` Parameters

Parameter	Default	Description
name	If the name is omitted, all applications are undeployed.	The name of the application.
dropTables	Value of the `drop-tables-at-undeploy` attribute in `sun-ejb-jar.xml`.	(optional) If `true`, and deployment and undeployment occur in the same JVM session, database tables that were automatically created when the bean(s) were deployed are dropped when the bean(s) are undeployed. If `true`, the *name* parameter must be specified or tables may not be dropped.

TABLE 6 `embedded-glassfish:undeploy` Parameters *(Continued)*

Parameter	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.

`embedded-glassfish:stop` Goal

This goal stops the server. You can set the parameters described in the following table.

TABLE 7 `embedded-glassfish:stop` Parameters

Parameter	Default	Description
serverID	`maven`	(optional) The ID of the server to stop.

Using the EJB 3.1 Embeddable API with Embedded Enterprise Server

The EJB 3.1 Embeddable API is designed for unit testing of EJB modules. You must use this API with a pre-installed, nonembedded Enterprise Server instance. However, you can take advantage of the embedded Enterprise Server's ease of use by referencing the nonembedded Enterprise Server instance with the `glassfish-embedded-static-shell.jar` file.

Embedded Enterprise Server is not related to the EJB 3.1 Embeddable API. Neither the Embedded Server API nor the Maven plug-in apply to embeddable EJB applications.

The EJB 3.1 Embeddable API is described in Java Specification Request (JSR) 318 (`http://jcp.org/en/jsr/detail?id=318`). An `ejb-embedded` sample is included in the samples available at Java EE 6 Downloads (`http://java.sun.com/javaee/downloads/index.jsp`) or Code Samples (`http://java.sun.com/javaee/reference/code/index.jsp`).

▼ To Use the EJB 3.1 Embeddable API with Embedded Enterprise Server

1 **To specify Enterprise Server as the Container Provider, include `glassfish-embedded-static-shell.jar` in the class path of your embeddable EJB application.**

See "Setting the Class Path" on page 14 and Section 22.3.3 of the EJB 3.1 Specification, *Embeddable Container Bootstrapping*.

2 **Configure any required resources.**

For more information about configuring resources, see the Administration Console Online Help or Part III, "Resources and Services Administration," in *Sun GlassFish Enterprise Server v3 Administration Guide*. The `jdbc/__default` Java DB database is preconfigured with all distributions of Enterprise Server.

3 **Invoke one of the `createEJBContainer` methods.**

Note – *Do not* deploy your embeddable EJB application or any of its dependent Java EE modules before invoking one of the `createEJBContainer` methods. These methods perform deployment in the background and do not load previously deployed applications or modules.

---

4 **To change the Instance Root Directory, set the `org.glassfish.ejb.embedded.glassfish.instance.root` system property value by using the `createEJBContainer(Map<?, ?> properties)` method.**

The default Instance Root Directory location is *as-install*`domains/domain1`. This system property applies only to embeddable EJB applications used with nonembedded Enterprise Server.

5 **Close the EJB container properly to release all acquired resources and threads.**

Changing Log Levels in Embedded Enterprise Server

To change log levels in Embedded Enterprise Server, you can follow the steps in this section or you can use the Embedded Server API as shown in Example 9. For more information about Enterprise Server logging, see Chapter 7, "Administering the Logging Service," in *Sun GlassFish Enterprise Server v3 Administration Guide*.

▼ To Change Log Levels in Embedded Enterprise Server

1. **Ensure that you have write permission access to the `$JAVA_HOME/jre/lib/logging.properties` file.**
2. **Use a text editor to edit the `$JAVA_HOME/jre/lib/logging.properties` file.**
3. **Change the `java.util.logging.ConsoleHandler.level` log level to `FINE` or `FINEST`.**
4. **Add Enterprise Server log levels to the end of the file and adjust them as necessary.**

   For example, add `javax.enterprise.system.tools.deployment.level=FINE`.

Restrictions for Embedded Enterprise Server

The `glassfish-embedded-web.jar` file for embedded Enterprise Server supports only these features of nonembedded Enterprise Server:

- The following web technologies of the Java EE platform:
  - Java Servlet API 2.5
  - JavaServer Pages™ (JSP™) technology 2.1
  - JavaServer™ Faces technology 1.0
- JDBC-technology connection pooling
- Java Persistence API 1.0
- Java Transaction API
- Java Transaction Service

The other embedded Enterprise Server JAR files, `glassfish-embedded-all.jar` and `glassfish-embedded-static-shell.jar`, support all features of nonembedded Enterprise Server with these exceptions:

- Installers
- Administration Console
- Update Tool
- Apache Felix OSGi framework
- EJB Timer Service
- The Maven plug-in for embedded Enterprise Server does not support application clients.
- Applications that require ports for communication, such as remote EJB components, do not work with the EJB 3.1 Embeddable API running with embedded Enterprise Server if a nonembedded Enterprise Server is running in parallel.

Embedded Enterprise Server requires no installation or configuration. As a result, the following files and directories are absent from the file system until embedded Enterprise Server is started:

- `default-web.xml` file
- `domain.xml` file
- Applications directory
- Instance root directory

When embedded Enterprise Server is started, the base installation directory that Enterprise Server uses depends on the options with which Enterprise Server is started. For more information, see "Specifying an Embedded Enterprise Server File System" on page 16. If necessary, embedded Enterprise Server creates a base installation directory. Embedded Enterprise Server then copies the following directories and their contents from the Java archive (JAR) file in which embedded Enterprise Server is distributed:

- `domains`
- `lib`

If necessary, Enterprise Server also creates an instance root directory.

Index