Eclipse GlassFish Embedded Server Guide, Release 7

Eclipse GlassFish

Embedded Server Guide

Release 7

Contributed 2018 - 2024

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

Note: The main thrust of the Eclipse GlassFish 7 release is to provide an application server for developers to explore and begin exploiting the new and updated technologies in the Jakarta EE 10 platform. Thus, the embedded server feature of Eclipse GlassFish was not a focus of this release. This feature is included in the release, but it may not function properly with some of the new features added in support of the Jakarta EE 10 platform.

Preface

This documentation is part of the Java Enterprise Edition contribution to the Eclipse Foundation and is not intended for use in relation to Java Enterprise Edition or Orace GlassFish. The documentation is in the process of being revised to reflect the new Jakarta EE branding. Additional changes will be made as requirements and procedures evolve for Jakarta EE. Where applicable, references to Jakarta EE or Java Enterprise Edition should be considered references to Jakarta EE.

Please see the Title page for additional license information.

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

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

Eclipse GlassFish 7 is developed through the GlassFish project open-source community at https://github.com/eclipse-ee4j/glassfish. The GlassFish project provides a structured process for developing the Eclipse GlassFish platform that makes the new features of the Jakarta EE platform available faster, while maintaining the most important feature of Jakarta EE: compatibility. It enables Java developers to access the Eclipse GlassFish source code and to contribute to the development of the Eclipse GlassFish.

The following topics are addressed here:

Eclipse GlassFish Documentation Set

The Eclipse GlassFish documentation set describes deployment planning and system installation. For an introduction to Eclipse GlassFish, refer to the books in the order in which they are listed in the following table.

Book Title	Description
Release Notes	Provides late-breaking information about the software and the documentation and includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK), and database drivers.

Book Title	Description
Quick Start Guide	Explains how to get started with the Eclipse GlassFish product.
Installation Guide	Explains how to install the software and its components.
Upgrade Guide	Explains how to upgrade to the latest version of Eclipse GlassFish. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Deployment Planning Guide	Explains how to build a production deployment of Eclipse GlassFish that meets the requirements of your system and enterprise.
Administration Guide	Explains how to configure, monitor, and manage Eclipse GlassFish 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.
Security Guide	Provides instructions for configuring and administering Eclipse GlassFish security.
Application Deployment Guide	Explains how to assemble and deploy applications to the Eclipse GlassFish and provides information about deployment descriptors.
Application Development Guide	Explains how to create and implement Java Platform, Enterprise Edition (Jakarta EE platform) applications that are intended to run on the Eclipse GlassFish. These applications follow the open Java standards model for Jakarta EE components and application programmer interfaces (APIs). This guide provides information about developer tools, security, and debugging.
Add-On Component Development Guide	Explains how to use published interfaces of Eclipse GlassFish to develop add-on components for Eclipse GlassFish. This document explains how to perform only those tasks that ensure that the add-on component is suitable for Eclipse GlassFish.
Embedded Server Guide	Explains how to run applications in embedded Eclipse GlassFish and to develop applications in which Eclipse GlassFish is embedded.
High Availability Administration Guide	Explains how to configure Eclipse GlassFish to provide higher availability and scalability through failover and load balancing.
Performance Tuning Guide	Explains how to optimize the performance of Eclipse GlassFish.
Troubleshooting Guide	Describes common problems that you might encounter when using Eclipse GlassFish and explains how to solve them.
Error Message Reference	Describes error messages that you might encounter when using Eclipse GlassFish.
Reference Manual	Provides reference information in man page format for Eclipse GlassFish administration commands, utility commands, and related concepts.

Book Title	Description
Message Queue Release Notes	Describes new features, compatibility issues, and existing bugs for Open Message Queue.
Message Queue Technical Overview	Provides an introduction to the technology, concepts, architecture, capabilities, and features of the Message Queue messaging service.
Message Queue Administration Guide	Explains how to set up and manage a Message Queue messaging system.
Message Queue Developer's Guide for JMX Clients	Describes the application programming interface in Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX).
Message Queue Developer's Guide for Java Clients	Provides information about concepts and procedures for developing Java messaging applications (Java clients) that work with Eclipse GlassFish.
Message Queue Developer's Guide for C Clients	Provides programming and reference information for developers working with Message Queue who want to use the C language binding to the Message Queue messaging service to send, receive, and process Message Queue messages.

Related Documentation

The following tutorials explain how to develop Jakarta EE applications:

- Your First Cup: An Introduction to the Jakarta EE Platform. For beginning Jakarta EE programmers, this short tutorial 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.
- The Jakarta EE Tutorial. This comprehensive tutorial explains how to use Jakarta EE platform technologies and APIs to develop Jakarta EE applications.

Javadoc tool reference documentation for packages that are provided with Eclipse GlassFish is available as follows.

- The Jakarta EE specifications and API specification is located at https://jakarta.ee/specifications/.
- The API specification for Eclipse GlassFish 7, including Jakarta EE platform packages and nonplatform packages that are specific to the Eclipse GlassFish product, is located at https://glassfish.org/docs/.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see the NetBeans Documentation, Training & Support page.

For information about the Derby database for use with the Eclipse GlassFish, see the Derby page.

The Jakarta EE Samples project is a collection of sample applications that demonstrate a broad

range of Jakarta EE technologies. The Jakarta EE Samples are bundled with the Jakarta EE Software Development Kit (SDK) and are also available from the repository (https://github.com/eclipse-ee4j/glassfish-samples).

Typographic Conventions

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

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.

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.

Symbol	Description	Example	Meaning
>	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.

Placeholder	Description	Default Value
as-install	Represents the base installation directory for Eclipse GlassFish. In configuration files, as-install is represented as follows: `${com.sun.aas.installRoot}`	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac OS operating system: user's-home-directory`/glassfish7/glassfish` • Installations on the Windows operating system: SystemDrive`:\glassfish7\glassfish`
as-install-parent	Represents the parent of the base installation directory for Eclipse GlassFish.	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac operating system: user's-home-directory`/glassfish7` • Installations on the Windows operating system: SystemDrive`:\glassfish7`
domain-root-dir	Represents the directory in which a domain is created by default.	as-install`/domains/`
domain-dir	Represents the directory in which a domain's configuration is stored. In configuration files, domain-dir is represented as follows: `${com.sun.aas.instanceRoot}`	domain-root-dir/domain-name
instance-dir	Represents the directory for a server instance.	domain-dir/instance-name

Eclipse GlassFish 7 Embedded Server Guide

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

The following topics are addressed here:

Introduction to Embedded Eclipse GlassFish

Embedded Eclipse GlassFish enables you to use Eclipse GlassFish as a library or run applications from command line. Embedded Eclipse GlassFish also enables you to run Eclipse GlassFish inside any Virtual Machine for the Java platform (Java Virtual Machine or JVM).

No installation or configuration of embedded Eclipse GlassFish is required. The ability to run Eclipse GlassFish without installation or configuration simplifies the process of bundling Eclipse GlassFish with applications or running applications from command line.

You can use embedded Eclipse GlassFish in the following ways:

- With the Embedded Server API (see Including the Eclipse GlassFish Embedded Server API in Applications)
- As an executable JAR, executed with `java -jar` (see Running Eclipse GlassFish Embedded Server from Command Line)
- With the Maven Plug-in (see Testing Applications with the Maven Plug-in for Embedded Eclipse GlassFish)
- With the EJB 3.1 Embeddable API (see Using the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish)

Embedded Eclipse GlassFish provides a plug-in for Apache Maven (`http://maven.apache.org/`). This plug-in simplifies the testing of applications by enabling you to build an application and run it with Eclipse GlassFish on a single system. Testing and debugging are simplified because the need for an installed and configured Eclipse GlassFish is eliminated. Therefore, you can run unit tests automatically in every build.

For information on how to embed Eclipse GlassFish in an OSGi environment, see the Eclipse GlassFish Add-On Component Development Guide.

Embedded Eclipse GlassFish Editions

Embedded Eclipse GlassFish comes with the following flavours:

`glassfish-embedded-all.jar`

Contains classes needed for deploying all Jakarta EE application types. Download this file from `https://central.sonatype.com/artifact/org.glassfish.main.extras/glassfish-embedded-all`.

`glassfish-embedded-web.jar`

Contains classes needed for deploying Jakarta EE web applications. Download this file from `https://central.sonatype.com/artifact/org.glassfish.main.extras/glassfish-embedded-web`.

`glassfish-embedded-static-shell.jar`

Contains references to classes needed for deploying Jakarta EE applications. Must be used with a non-embedded installation of Eclipse GlassFish. It can run the same Jakarta EE applications that are supported by the non-embedded installation that contains it. Reference this file from the as-install`/lib/embedded` directory of a nonembedded Eclipse GlassFish installation. Do not move this file or it will not work. For an explanation of as-install, see Installation Root Directory.

Embedded Eclipse GlassFish File System

The following Embedded Eclipse GlassFish directories and files are important if you are referencing a nonembedded installation of Eclipse GlassFish:

- Installation Root Directory
- Instance Root Directory
- The `domain.xml` File

Installation Root Directory

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

- `domains`
- `lib`

Specify the installation root directory only if you have a valid nonembedded Eclipse GlassFish installation and are using `glassfish-embedded-static-shell.jar`.

Instance Root Directory

The instance root directory, also known as the domain directory, represented as domain-dir, is the parent directory of a server instance directory. Embedded Eclipse GlassFish uses the server instance directory for domain configuration files.

If you have valid custom as-install and domain-dir directories, specify both in the `BootstrapProperties` and `GlassFishProperties` classes respectively as described in Creating and Configuring an Embedded Eclipse GlassFish.

If domain-dir is not specified, Eclipse GlassFish creates a directory named `gfembed`random-number`tmp` in a temporary directory, where random-number is a randomly generated 19-digit number. Eclipse GlassFish then copies configuration files into this directory. The temporary directory is the value of the system property `java.io.tmpdir`. You can override this value by specifying the `glassfish.embedded.tmpdir` property in the `GlassFishProperties` class or as a system property.

The `domain.xml` File

Using an existing `domain.xml` file avoids the need to configure embedded Eclipse GlassFish 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 Eclipse GlassFish. To specify an existing `domain.xml` file, invoke the `setConfigFileURI` method of the `GlassFishProperties` class as described in Creating and Configuring an Embedded Eclipse GlassFish.

The built-in `domain.xml` file used by default by Embedded Eclipse GlassFish can be found in the Embedded Eclipse GlassFish JAR files, in path `org/glassfish/embed/domain.xml`. You can customize this file and pass it in using the `setConfigFileURI` method while creating an Embedded Eclipse GlassFish.

Running Eclipse GlassFish Embedded Server from Command Line

Embedded Eclipse GlassFish JAR file can be started as an executable JAR from command line, such as:

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

```
java -jar glassfish-embedded-web.jar
```

```
java -jar ${as-install}/lib/embedded/glassfish-embedded-static-shell.jar
```

For simplicity, this guide will use `glassfish-embedded.jar` to reference any of the above JAR files.

For description of all the JAR files, see Embedded Eclipse GlassFish editions

This starts the server and configures it according to the given arguments or configuration files.

If executed without arguments or if no application is deployed, the server is started and waits for further admin commands in a loop. The server can be terminated with the `exit` or `quit` commands or pressing Ctrl+C.

If `glassfish.properties` file exists in the current directory, properties in this file are set as if the `--properties=glassfish.properties` argument was defined on the command line.

If `autodeploy` directory exists in the current directory, files and directories in that directory are deployed as applications on startup.

If `glassfish-domain` directory exists in the current directory, it will be used as the domain directory unless specified explicitly with `--domainDir=DIRECTORY`

If `domain.xml` file exists in the current directory, it will be used as the domain configuration file unless specified explicitly with `--domainConfigFile=FILE`

By default, Eclipse GlassFish Embedded is started with the following configuration:

- HTTP listener enabled on port 8080
- HTTPS listener disabled

It's also possible to start Eclipse GlassFish on command line with an alternative way, via the main class:

```
    java -cp glassfish-embedded.jar org.glassfish.runnablejar.UberMain
```

Nnote that this way of starting will require some `--add-opens` and `--add-exports` Java arguments)

Examples of Running Embedded Eclipse GlassFish from Command Line

Example 1: Run an application from command line

On port 8080 and root context by default:

```
java -jar glassfish-embedded.jar app.war
```

Example 2: Run an app on a different port

```
java -jar glassfish-embedded.jar --httpPort=8090 app.war
```

Example 3: Run custom commands on startup

Deploys an application with a custom root context and prints info about the deployed application. Custom commands need to be enclosed in quotes if they contain spaces.

```
java -jar glassfish-embedded.jar "deploy --contextroot=/app app.war" list-applications
```

Example 4: Run 2 applications from command line

Deploys applications on different context roots, based on the file name or info in the application descriptors.

```
java -jar glassfish-embedded-all.jar app1.war app2.war
```

Command-line Arguments Supported by Embedded Eclipse GlassFish

```
java -jar glassfish-embedded-all.jar [--properties=FILE]
    [-p=PORT, --httpPort=PORT, --port=PORT] [--httpsPort=PORT_NUMBER]
    [--domainConfigFile=FILE]
    [--domainDir=DIRECTORY, --instanceRoot=DIRECTORY]
    [--noListener, --noPort] [--autoDeployDir=DIRECTORY] [--logLevel=LEVEL]
    [--logProperties=FILE] [--noInfo] [--shut-down, --shutdown, --stop]
    [--help] [applications or admin commands...]
```

Command line options with a value can be specified in two forms: `--option=value` (e.g.,

`--httpPort=8090`) or `--option value` (e.g., `--httpPort 8090`).

`--properties=FILE`

Load properties from a file. This option can be repeated to load properties from multiple files. The propertes in the file can be any of the following:

- Any properties supported by Embedded $Eclipse GlassFish. See Configuration properties supported by Embedded Eclipse GlassFish.
- Any command line options with the name of the option as the key, without the initial hyphens, and the value of the option as the value.
- Keys that start with the "command." prefix, followed by any text. The value will be treated as a command to execute at startup.
- Keys that start with the "deploy." prefix, followed by any text. The value will be treated as an application to deploy at startup, as if it was specified on the command line.
- If a property name doesn't match any already supported patterns and is not a recognized GlassFish property, it will be set as a system property, if it's not already defined. For example, the $Eclipse GlassFish domain directory can be specified with the usual Embedded $Eclipse GlassFish property "glassfish.embedded.tmpdir=myDomainDir", as well as with the property "domainDir=myDomainDir" that represents the "--domainDir=myDomainDir" command-line option. A command to deploy an application can be specified via a property key "command.deploy.app=deploy --contextroot=app myapp.war". An application to deploy at startup with the default deploy behavior can be specified via a property key "deploy.app=myapp.war". The property "properties" can also be defined in this file, pointing to another file. In that case, properties will be loaded also from that file.

`-p=PORT, --httpPort=PORT, --port=PORT`

Bind the HTTP listener to the specified port. If not set, the HTTP listener binds to port 8080 by default, unless it's disabled by the --noListener argument.

`--httpsPort=PORT_NUMBER`

Bind the HTTPS listener to the specified port. If not set, the HTTPS listener is disabled by default.

`--domainConfigFile=FILE`

Set the location of domain configuration file (i.e., domain.xml) using which Embedded Eclipse GlassFish should run.

`--domainDir=DIRECTORY, --instanceRoot=DIRECTORY`

Set the instance root (a.k.a. domain dir) using which Embedded Eclipse GlassFish should run.

`--noListener, `--noPort`

Disable the HTTP listener, which is by default enabled and bound to port 8080.

`--autoDeployDir=DIRECTORY`

Files and directories in this directory will be deployed as applications (in random order), as if they

were specified on the command line. The default directory name is 'autodeploy'.

`--logLevel=LEVEL`

Set the log level of all loggers to LEVEL

`--logProperties=FILE`

Set logging properties from file FILE

`--noInfo`

Disable printing information about deployed applications after startup

`--shut-down, --shutdown, --stop`

Shut down GlassFish and the whole JVM process after server is started and initialized. This is useful to start the server, perform some action during startup (e.g. during application deployment), and shut down the application cleanly. Also useful for Class Data Sharing and similar startup optimizations - to start the server, get it to a ready state (applications deployed, etc.), and then shut down cleanly, so that the JVM can store the cached data.

`--prompt`

Run interactive prompt that allows running admin commands. This is useful in development to manipulate a running GlassFish instance. After exiting the prompt, the server shuts down.

`--help`

Print help information

Any argument that doesn't start with a hyphen (-), is treated as follows:

- If it's a file or directory, it's deployed at startup as an application. If it's the only application deployed at startup, it's deployed under the root context '/'. Otherwise it's deployed under the context root derived from the name of the file or deployment descriptors.
- In all other cases, the argument is executed as a Eclipse GlassFish admin command. Individual commands must be enclosed in quotes if they contain spaces. Eclipse GlassFish admin commands are the same commands supported by Eclipse GlassFish Server's "asadmin" command line tool or by the "CommandRunner" Java class in the Eclipse GlassFish Simple Public API.

Including the Eclipse GlassFish Embedded Server API in Applications

Eclipse GlassFish provides an application programming interface (API) for developing applications in which Eclipse GlassFish is embedded. For details, see the `org.glassfish.embeddable` packages at `https://www.javadoc.io/doc/org.glassfish.main.common/simple-glassfish-api/latest/index.html`.

The following topics are addressed here:

- Setting the Class Path
- Creating, Starting, and Stopping Embedded Eclipse GlassFish
- Deploying and Undeploying an Application in an Embedded Eclipse GlassFish
- Running `asadmin` Commands Using the Eclipse GlassFish Embedded Server API
- Sample Applications

Setting the Class Path

To enable your applications to locate the class libraries for embedded Eclipse GlassFish, add a JAR file corresponding to one of the Embedded Eclipse GlassFish editions to your class path.

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 Eclipse GlassFish

Before you can run applications, you must set up and run the embedded Eclipse GlassFish.

The following topics are addressed here:

- Creating and Configuring an Embedded Eclipse GlassFish
- Running an Embedded Eclipse GlassFish

Creating and Configuring an Embedded Eclipse GlassFish

To create and configure an embedded Eclipse GlassFish, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.BootstrapProperties` class.
2. Invoke any methods for configuration settings that you require. This is optional.
3. Invoke the `GlassFishRuntime.bootstrap()` or `GlassFishRuntime.bootstrap(BootstrapProperties)` method to create a `GlassFishRuntime` object.
4. Instantiate the `org.glassfish.embeddable.GlassFishProperties` class.

5. Invoke any methods for configuration settings that you require. This is optional.
6. Invoke the `glassfishRuntime.newGlassFish(GlassFishProperties)` method to create a `GlassFish` object.

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

Table 1-1 Methods of the `BootstrapProperties` Class

Purpose	Method	Default Value
References an existing Installation Root Directory, also called as-install	`setInstallRoot(String as-install)`	None. If `glassfish-embedded-static-shell.jar` is used, the Installation Root Directory is automatically determined and need not be specified.

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

Table 1-2 Methods of the `GlassFishProperties` Class

Purpose	Method	Default Value
References an existing Instance Root Directory, also called domain-dir	`setInstanceRoot(String domain-dir)`	In order of precedence: • `glassfish.embedded.tmpdir` property value specified in `GlassFishProperties` object • `glassfish.embedded.tmpdir` system property value • `java.io.tmpdir` system property value • as-install`/domains/domain1` if a nonembedded installation is referenced
Creates a new or references an existing configuration file	`setConfigFileURI(String configFileURI)`	In order of precedence: • domain-dir`/config/domain.xml` if domain-dir was set using `setInstanceRoot` • built-in embedded `domain.xml`
Specifies whether the configuration file is read-only	`setConfigFileReadOnly(boolean readOnly)`	`true`

Purpose	Method	Default Value
Sets the port on which Embedded Eclipse GlassFish listens.	`setPort`(String networkListener, int port)	none

Do not use `setPort` if you are using `setInstanceRoot` or `setConfigFileURI`.

Example 1-1 Creating an Embedded Eclipse GlassFish

This example shows code for creating an Embedded Eclipse GlassFish.

```
...
import org.glassfish.embeddable.*;
...
    GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish();
    glassfish.start();
...
```

Example 1-2 Creating an Embedded Eclipse GlassFish with configuration customizations

This example shows code for creating an Embedded Eclipse GlassFish using the existing domain-dir `C:\samples\test\applicationserver\domains\domain1`.

```
// ...
import org.glassfish.embeddable.*;
    // ...
    BootstrapProperties bootstrapProperties = new BootstrapProperties();
    bootstrapProperties.setInstallRoot("C:\\samples\\test\\applicationserver");
    GlassFishRuntime glassfishRuntime = GlassFishRuntime.bootstrap(
bootstrapProperties);

    GlassFishProperties glassfishProperties = new GlassFishProperties();
    glassfishProperties.setInstanceRoot("C:\\samples\\test\\applicationserver\\
domains\\domain1");
    GlassFish glassfish = glassfishRuntime.newGlassFish(glassfishProperties);

    glassfish.start();
    // ...
```

Configuration properties supported by Embedded Eclipse GlassFish

In `GlassFishProperties` and in a properties file, Embedded Eclipse GlassFish supports the same configuration properties as the `set` and `get` administration commands of the Eclipse GlassFish Server.

In addition, it also accepts properties with the `embedded-glassfish-config.` prefix. This prefix is removed before applying the property (e.g., `resources.jdbc-connection-pool…` can be defined as

`embedded-glassfish-config.resources.jdbc-connection-pool⋯`). This prefix is no longer necessary and its usage is deprecated, but it's supported for backwards compatibility.

Running Embedded Eclipse GlassFish from a Java application

After you create an embedded Eclipse GlassFish server as described in Creating and Configuring an Embedded Eclipse GlassFish, you can perform operations such as:

- Setting the Port of an Embedded Eclipse GlassFish From an Application
- Starting an Embedded Eclipse GlassFish From an Application
- Stopping an Embedded Eclipse GlassFish From an Application

Setting the Port of an Embedded Eclipse GlassFish From an Application

You must set the server's HTTP or HTTPS 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.

Do not use `setPort` if you are using `setInstanceRoot` or `setConfigFileURI`. These methods set the port indirectly.

- To set the port directly, invoke the `setPort` method of the `GlassFishProperties` object.
- To set the port indirectly, use a `domain.xml` file that sets the port. For more information, see The `domain.xml` File.

Example 1-3 Setting the port of an Embedded Eclipse GlassFish

This example shows code for setting the port of an embedded Eclipse GlassFish.

```
...
import org.glassfish.embeddable.*;
...
    GlassFishProperties glassfishProperties = new GlassFishProperties();
    glassfishProperties.setPort("http-listener", 8080);
    glassfishProperties.setPort("https-listener", 8181);
...
```

Starting an Embedded Eclipse GlassFish From an Application

To start an embedded Eclipse GlassFish, invoke the `start` method of the `GlassFish` object.

Example 1-4 Starting an Embedded Eclipse GlassFish

This example shows code for setting the port and starting an embedded Eclipse GlassFish. This example also includes the code from Example 1-1 for creating a `GlassFish` object.

```
...
import org.glassfish.embeddable.*;
...
```

```
    GlassFishProperties glassfishProperties = new GlassFishProperties();
    glassfishProperties.setPort("http-listener", 8080);
    glassfishProperties.setPort("https-listener", 8181);
    ...
    GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish
(glassfishProperties);
    glassfish.start();
...
```

Stopping an Embedded Eclipse GlassFish From an Application

The API for embedded Eclipse GlassFish 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 Eclipse GlassFish, invoke the `stop` method of an existing `GlassFish` object.

Example 1-5 Stopping an Embedded Eclipse GlassFish

This example shows code for prompting the user to press the Enter key to stop an embedded Eclipse GlassFish. Code for creating a `GlassFish` object is not shown in this example. For an example of code for creating a `GlassFish` object, see Example 1-1.

```
...
import java.io.BufferedReader;
...
import org.glassfish.embeddable.*;
...
    System.out.println("Press Enter to stop server");
        // wait for Enter
    glassfish.stop(); // Stop Embedded GlassFish
...
```

As an alternative, you can use the `dispose` method to stop an embedded Eclipse GlassFish and dispose of the temporary file system.

Deploying and Undeploying an Application in an Embedded Eclipse GlassFish

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

The following topics are addressed here:

- To Deploy an Application From an Archive File or a Directory
- Undeploying an Application
- Creating a Scattered Archive
- Creating a Scattered Enterprise Archive

For general information about deploying applications in Eclipse GlassFish, see the Eclipse GlassFish 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 Jakarta EE specifications define for the type of archive that the file contains. For more information, see "Deploying Applications" in Eclipse GlassFish 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 Jakarta EE archive file as laid out by the Eclipse GlassFish. 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 Eclipse GlassFish Application Deployment Guide.

If some of the resources needed by an application are not under the application's directory, see Creating a Scattered Archive.

1. Instantiate the `java.io.File` class to represent the archive file or directory.
2. Invoke the `getDeployer` method of the `GlassFish` object to get an instance of the `org.glassfish.embeddable.Deployer` class.
3. Invoke the `deploy(File archive, String… params)` method of the instance of the `Deployer` object.
   Specify the `java.io.File` class instance you created previously as the first method parameter.
   For information about optional parameters you can set, see the descriptions of the `deploy(1)` subcommand parameters. Simply quote each parameter in the method, for example `"--force=true"`.

Example 1-6 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 name, contextroot, and force parameters. This example also includes the code from Example 1-1 for creating `GlassFishProperties` and `GlassFish` objects.

```
...
import java.io.File;
...
import org.glassfish.embeddable.*;
...
    GlassFishProperties glassfishProperties = new GlassFishProperties();
```

```
    glassfishProperties.setPort("http-listener", 8080);
    glassfishProperties.setPort("https-listener", 8181);
    ...
    GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish
(glassfishProperties);
    glassfish.start();
    File war = new File("c:\\samples\\simple.war");
    Deployer deployer = glassfish.getDeployer();
    deployer.deploy(war, "--name=simple", "--contextroot=simple", "--force=true");
    // deployer.deploy(war) can be invoked instead. Other parameters are optional.
...
```

Undeploying an Application

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

If you reference a nonembedded Eclipse GlassFish installation using the `glassfish-embedded-static-shell.jar` file and do not undeploy your applications in the same server life cycle in which you deployed them, expanded archives for these applications remain under the domain-dir`/applications` directory.

To undeploy an application, invoke the `undeploy` method of an existing `Deployer` object. In the method invocation, pass the name of the application as a parameter. This name is specified when the application is deployed.

For information about optional parameters you can set, see the descriptions of the `deploy`(1) command parameters. Simply quote each parameter in the method, for example `"--cascade=true"`.

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

Example 1-7 Undeploying an Application

This example shows code for undeploying the application that was deployed in Example 1-6.

```
...
import org.glassfish.embeddable.*;
...
    deployer.undeploy(war, "--droptables=true", "--cascade=true");
...
```

Creating a Scattered Archive

Deploying a module from a scattered archive (WAR or JAR) enables you to deploy an unpackaged module whose resources, deployment descriptor, and classes are in any location. Deploying a module from a scattered archive simplifies the testing of a module during development, especially

if all the items that the module requires are not available to be packaged.

In a scattered archive, these items are not required to be organized in a specific directory structure. Therefore, you must specify the location of the module's resources, deployment descriptor, and classes when deploying the module.

To create a scattered archive, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.archive.ScatteredArchive` class.
2. Invoke the `addClassPath` and `addMetadata` methods if you require them.
3. Invoke the `toURI` method to deploy the scattered archive.

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

Table 1-3 Constructors and Methods of the `ScatteredArchive` Class

Purpose	Method	Default Value
Creates and names a scattered archive	`ScatteredArchive(String name, ScatteredArchive.Type type)`	None
Creates and names a scattered archive based on a top-level directory. If the entire module is organized under the topDir, this is the only method necessary. The topDir can be null if other methods specify the remaining parts of the module.	`ScatteredArchive(String name, ScatteredArchive.Type type, File topDir)`	None
Adds a directory to the classes classpath	`addClassPath(File path)`	None
Adds a metadata locator	`addMetaData(File path)`	None
Adds and names a metadata locator	`addMetaData(File path, String name)`	None
Gets the deployable URI for this scattered archive	`toURI()`	None

Example 1-8 Deploying an Application From a Scattered Archive

This example shows code for creating a WAR file and using the `addClassPath` and `addMetadata` methods. This example also includes the code from Example 1-6 for deploying an application from an archive file.

```
...
import java.io.File;
...
import org.glassfish.embeddable.*;
...
    GlassFishProperties glassfishProperties = new GlassFishProperties();
    glassfishProperties.setPort("http-listener", 9090);
    GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish
(glassfishProperties);
    glassfish.start();
    Deployer deployer = glassfish.getDeployer();
    ScatteredArchive archive = new ScatteredArchive("testapp", ScatteredArchive.Type
.WAR);
    // target/classes directory contains complied servlets
    archive.addClassPath(new File("target", "classes"));
    // resources/sun-web.xml is the WEB-INF/sun-web.xml
    archive.addMetadata(new File("resources", "sun-web.xml"));
    // resources/web.xml is the WEB-INF/web.xml
    archive.addMetadata(new File("resources", "web.xml"));
    // Deploy the scattered web archive.
    String appName = deployer.deploy(archive.toURI(), "--contextroot=hello");

    deployer.undeploy(appName);
    glassfish.stop();
    glassfish.dispose();
...
```

Creating a Scattered Enterprise Archive

Deploying an application from a scattered enterprise archive (EAR) enables you to deploy an unpackaged application whose resources, deployment descriptor, and classes are in any location. Deploying an application from a scattered archive simplifies the testing of an application during development, especially if all the items that the application requires are not available to be packaged.

In a scattered archive, these items are not required to be organized in a specific directory structure. Therefore, you must specify the location of the application's resources, deployment descriptor, and classes when deploying the application.

To create a scattered enterprise archive, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.archive.ScatteredEnterpriseArchive` class.
2. Invoke the `addArchive` and `addMetadata` methods if you require them.
3. Invoke the `toURI` method to deploy the scattered enterprise archive.

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

Table 1-4 Constructors and Methods of the `ScatteredEnterpriseArchive` Class

Purpose	Method	Default Value
Creates and names a scattered enterprise archive	`ScatteredEnterpriseArchive(String name)`	None
Adds a module or library	`addArchive(File archive)`	None
Adds a module or library	`addArchive(File archive, String name)`	None
Adds a module or library	`addArchive(URI URI)`	None
Adds a module or library	`addArchive(URI URI, String name)`	None
Adds a metadata locator	`addMetaData(File path)`	None
Adds and names a metadata locator	`addMetaData(File path, String name)`	None
Gets the deployable URI for this scattered archive	`toURI()`	None

Example 1-9 Deploying an Application From a Scattered Enterprise Archive

This example shows code for creating an EAR file and using the `addArchive` and `addMetadata` methods. This example also includes code similar toExample 1-8 for creating a scattered archive.

```
...
import java.io.File;
...
import org.glassfish.embeddable.*;
...
    GlassFishProperties glassfishProperties = new GlassFishProperties();
    glassfishProperties.setPort("http-listener", 9090);
    GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish
(glassfishProperties);
    glassfish.start();
    Deployer deployer = glassfish.getDeployer();

    // Create a scattered web application.
    ScatteredArchive webmodule = new ScatteredArchive("testweb", ScatteredArchive.
Type.WAR);
    // target/classes directory contains my complied servlets
```

```
    webmodule.addClassPath(new File("target", "classes"));
    // resources/sun-web.xml is my WEB-INF/sun-web.xml
    webmodule.addMetadata(new File("resources", "sun-web.xml"));

    // Create a scattered enterprise archive.
    ScatteredEnterpriseArchive archive = new ScatteredEnterpriseArchive("testapp");
    // src/application.xml is my META-INF/application.xml
    archive.addMetadata(new File("src", "application.xml"));
    // Add scattered web module to the scattered enterprise archive.
    // src/application.xml references Web module as "scattered.war".
    //Hence specify the name while adding the archive.
    archive.addArchive(webmodule.toURI(), "scattered.war");
    // lib/mylibrary.jar is a library JAR file.
    archive.addArchive(new File("lib", "mylibrary.jar"));
    // target/ejbclasses contain my compiled EJB module.
    // src/application.xml references EJB module as "ejb.jar".
    //Hence specify the name while adding the archive.
    archive.addArchive(new File("target", "ejbclasses"), "ejb.jar");

    // Deploy the scattered enterprise archive.
    String appName = deployer.deploy(archive.toURI());

    deployer.undeploy(appName);
    glassfish.stop();
    glassfish.dispose();
...
```

Running `asadmin` Commands Using the Eclipse GlassFish Embedded API

Running `asadmin` commands from an application enables the application to configure the embedded Eclipse GlassFish 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 Eclipse GlassFish, see the Eclipse GlassFish Administration Guide. For detailed information about `asadmin` commands, see Section 1 of the Eclipse GlassFish Reference Manual.

Ensure that your application has started an embedded Eclipse GlassFish before the application attempts to run `asadmin` commands. For more information, see Running an Embedded Eclipse GlassFish.

The `org.glassfish.embeddable` 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 1-10 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 1-4.

```
...
import org.glassfish.embeddable.*;
...
    String command = "create-jdbc-resource";
    String poolid = "--connectionpoolid=DerbyPool";
    String dbname = "jdbc/DerbyPool";
    CommandRunner commandRunner = glassfish.getCommandRunner();
    CommandResult commandResult = commandRunner.run(command, poolid, dbname);
...
```

Example 1-11 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 1-4.

```
...
import org.glassfish.embeddable.*;
...
    String command = "set-log-level";
    String weblevel = "jakarta.enterprise.system.container.web=FINE";
    CommandRunner commandRunner = glassfish.getCommandRunner();
    CommandResult commandResult = commandRunner.run(command, weblevel);
...
```

For another way to change log levels, see Changing Log Levels in Embedded Eclipse GlassFish.

Sample Applications

Example 1-12 Using an Existing `domain.xml` File and Deploying an Application From an Archive File

This example shows code for the following:

- 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`.

```
import java.io.File;
import java.io.BufferedReader;
import org.glassfish.embeddable.*;

public class Main {
```

```
     /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        File configFile = new File ("c:\\myapp\\embeddedserver\\domains\\domain1
\\config\\domain.xml");
        File war = new File("c:\\samples\\simple.war");
        try {
            GlassFishRuntime glassfishRuntime = GlassFishRuntime.bootstrap();
            ...
            GlassFishProperties glassfishProperties = new GlassFishProperties();
            glassfishProperties.setConfigFileURI(configFile.toURI());
            glassfishProperties.setConfigFileReadOnly(false);
            ...
            GlassFish glassfish = glassfishRuntime.newGlassFish(glassfishProperties);
            glassfish.start();

            Deployer deployer = glassfish.getDeployer();
            deployer.deploy(war, "--force=true");
        }
        catch (Exception e) {
            e.printStackTrace();
        }

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

Testing Applications with the Maven Plug-in for Embedded Eclipse GlassFish

If you are using Apache Maven (`http://maven.apache.org/`), the plug-in for embedded Eclipse GlassFish simplifies the testing of applications. This plug-in enables you to build and start an unpackaged application with a single Maven goal.

The following topics are addressed here:

- To Set Up Your Maven Environment
- To Build and Start an Application From Maven
- To Stop Embedded Eclipse GlassFish
- To Redeploy an Application That Was Built and Started From Maven
- Maven Goals for Embedded Eclipse GlassFish

Predefined Maven goals for embedded Eclipse GlassFish are described in Maven Goals for Embedded Eclipse GlassFish.

To use Maven with Embedded Eclipse GlassFish and the EJB 3.1 Embeddable API, see Using Maven with the EJB 3.1 Embeddable API and Embedded Eclipse GlassFish.

To Set Up Your Maven Environment

Setting up your Maven environment enables Maven to download the required embedded Eclipse GlassFish 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 (`http://maven.apache.org/`) is installed.

1. Identify the Maven plug-in for embedded Eclipse GlassFish.

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

```
...
        ...
        <plugins>
            ...
            <plugin>
                <groupId>org.glassfish.embedded</groupId>
                <artifactId>embedded-glassfish-maven-plugin</artifactId>
                <version>version</version>
            </plugin>
            ...
```

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

version

The version to use. The version of the final promoted build for this release is `7.0`. The Maven plug-in is not bound to a specific version of Eclipse GlassFish. You can specify the version you want to use. If no version is specified, a default version is used.

2. Configure the the path to the application WAR, and other standard settings.

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

```
...
        <plugins>
            ...
            <plugin>
                ...
                <configuration>
                    <app>target/test.war</app>
                    <port>8080</port>
                    <contextRoot>test</contextRoot>
                    <autoDelete>true</autoDelete>
                    ...
                </configuration>
                ...
            </plugin>
            ...
        </plugins>
...
```

app

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 Eclipse GlassFish.

3. Perform advanced plug-in configuration. This step is optional. Add the following `configuration` element to your POM file:

```
...
        <plugins>
            ...
            <plugin>
                ...
                <configuration>
                    <app>target/test.war</app>
                    <name>test</name>
                    <contextRoot>test</contextRoot>
                    <ports>
```

```
                    <http-listener>8080</http-listener>
                    <https-listener>8181</https-listener>
                </ports>
                <bootstrapProperties>
                    <property>test_key=test_value</property>
                </bootstrapProperties>
                <bootstrapPropertiesFile>
bootstrap.properties</bootstrapPropertiesFile>
                <glassfishProperties>
<property>server.jms-service.jms-host.default_JMS_host.port=17676</property>
                </glassfishProperties>
                <glassfishPropertiesFile>
glassfish.properties</glassfishPropertiesFile>
                <systemProperties>
                    <property>ANTLR_USE_DIRECT_CLASS_LOADING=true</property>
                </systemProperties>
                <systemPropertiesFile>system.properties</systemPropertiesFile>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>start</goal>
                        <goal>deploy</goal>
                        <goal>undeploy</goal>
                        <goal>stop</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        ...
    </plugins>
...
```

4. 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 Eclipse GlassFish.

Example 1-13 POM File for Configuring Maven to Use Embedded Eclipse GlassFish

This example shows a POM file for configuring Maven to use embedded Eclipse GlassFish.

```
<?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.example</groupId>
  <artifactId>maven-glassfish-plugin-tester</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <name>Maven Embedded Glassfish Plugin Example</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.glassfish.embedded</groupId>
        <artifactId>embedded-glassfish-maven-plugin</artifactId>
        <version>7.0</version>
        <configuration>
          <app>target/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>
</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 Eclipse GlassFish.

Before You Begin

Ensure that your Maven environment is configured, as described in To Set Up Your Maven Environment.

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.
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 Eclipse GlassFish.
   - Deploys your application.

     The application continues to run in Embedded Eclipse GlassFish until Embedded Eclipse GlassFish is stopped.

To Stop Embedded Eclipse GlassFish

1. Change to the root directory of the Maven project for your application.
2. Run the Maven goal to stop the application in embedded Eclipse GlassFish.

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

   This runs the `stop` method of the `GlassFish` object and any other methods that are required to shut down the server in an orderly fashion. See Stopping an Embedded Eclipse GlassFish From an Application.

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 Eclipse

GlassFish until Embedded Eclipse GlassFish is stopped. While the application is running, you can test changes to the application by redeploying it.

To redeploy, in the window from where the application was built and started from Maven, press Enter.

Maven Goals for Embedded Eclipse GlassFish

You can use the following Maven goals to test your applications with embedded Eclipse GlassFish:

- `embedded-glassfish:run` Goal
- `embedded-glassfish:start` Goal
- `embedded-glassfish:deploy` Goal
- `embedded-glassfish:undeploy` Goal
- `embedded-glassfish:stop` Goal
- `embedded-glassfish:admin` Goal

`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 1-5 `embedded-glassfish:run` Parameters

Parameter	Default	Description
app	None	The archive file or directory for the application to be deployed.
serverID	`maven`	(optional) The ID of the server to start.
containerType	`all`	(optional) The container to start: `web`, `ejb`, `jpa`, or `all`.
installRoot	None	(optional) The Installation Root Directory.

Parameter	Default	Description
instanceRoot	In order of precedence: • `glassfish.embedded.tmpdir` property value specified in `GlassFishProperties` object • `glassfish.embedded.tmpdir` system property value • `java.io.tmpdir` system property value • as-install`/domains/domain1` if a nonembedded installation is referenced	(optional) The Instance Root Directory
configFile	domain-dir`/config/domain.xml`	(optional) The configuration file.
port	None. Must be set explicitly or defined in the configuration file.	The HTTP or HTTPS port.
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 Eclipse GlassFish 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.
dbVendorName	None	(optional) The name of the database vendor for which tables can be created. Allowed values are `javadb`, `db2`, `mssql`, `mysql`, `oracle`, `postgresql`, `pointbase`, `derby` (also for CloudScape), and `sybase`, case-insensitive.
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.

Parameter	Default	Description
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.
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 Eclipse GlassFish installation.

`embedded-glassfish:start` Goal

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

Table 1-6 `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	None	(optional) The Installation Root Directory.
instanceRoot	In order of precedence: • `glassfish.embedded.tmpdir` system property value • `java.io.tmpdir` system property value • as-install`/domains/domain1`	(optional) The Instance Root Directory
configFile	domain-dir`/config/domain.xml`	(optional) The configuration file.
port	None. Must be set explicitly or defined in the configuration file.	The HTTP or HTTPS port.

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 Eclipse GlassFish 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 1-7 `embedded-glassfish:deploy` Parameters

Parameter	Default	Description
app	None	The archive file or directory for the application to be deployed.
serverID	`maven`	(optional) The ID of the server to start.
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 Eclipse GlassFish 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.
dbVendorName	None	(optional) The name of the database vendor for which tables can be created. Allowed values are `javadb`, `db2`, `mssql`, `oracle`, `postgresql`, `pointbase`, `derby` (also for CloudScape), and `sybase`, case-insensitive.

Parameter	Default	Description
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

If you reference a nonembedded Eclipse GlassFish installation using the `glassfish-embedded-static-shell.jar` file and do not undeploy your applications in the same server life cycle in which you deployed them, expanded archives for these applications remain under the domain-dir`/applications` directory.

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

Table 1-8 `embedded-glassfish:undeploy` Parameters

Parameter	Default	Description
name	If the name is omitted, all applications are undeployed.	The name of the application.
serverID	`maven`	(optional) The ID of the server to start.
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.
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 1-9 `embedded-glassfish:stop` Parameters

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

`embedded-glassfish:admin` Goal

This goal runs a Eclipse GlassFish administration command. You must use either the command and commandParameters parameters in combination or the commandLine parameter. For more information about administration commands, see the Eclipse GlassFish Reference Manual. You can set the parameters described in the following table.

Table 1-10 `embedded-glassfish:start` Parameters

Parameter	Default	Description
serverID	`maven`	(optional) The ID of the server on which to run the command.
command	None	The name of the command, for example `createJdbcResource`.
commandParameters	None	A map of the command parameters. See the `org.glassfish.embeddable.admin.CommandParameters` class at `https://www.javadoc.io/doc/org.glassfish.main.common/glassfish-api/latest/org/glassfish/api/admin/CommandParameters.html`.
commandLine	None	The full `asadmin` syntax of the command.

Using the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish

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

Embedded Eclipse GlassFish is not related to the EJB 3.1 Embeddable API, but you can use these APIs together.

The Maven plug-in does not apply to embeddable EJB applications. However, you can use Maven with the POM file shown in Using Maven with the EJB 3.1 Embeddable API and Embedded Eclipse GlassFish.

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` example is available at Code Samples (`https://jakarta.ee/learn/docs/jakartaee-tutorial/9.1/entbeans/ejb-embedded/ejb-embedded.html`).

The EJB 3.1 Embeddable API supports all EJB 3.1 Lite features with addition of the EJB timer service and testing of EJB modules packaged in a WAR file.

For EJB modules in a WAR file (or an exploded directory), if a web application has one EJB module, and there are no other EJB modules in the classpath, those entries (libraries) are ignored. If there are other EJB modules, a temporary EAR file is created. For EJB modules in a WAR file to be tested, the client code must use EJB modules with interfaces or without annotations. Those EJB modules are not part of the classpath and can't be loaded by the client class loader.

The following topics are addressed here:

- To Use the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish
- EJB 3.1 Embeddable API Properties
- Using Maven with the EJB 3.1 Embeddable API and Embedded Eclipse GlassFish

To Use the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish

1. To specify Eclipse GlassFish as the Container Provider, include `glassfish-embedded-static-shell.jar` or `glassfish-embedded-all.jar` in the class path of your embeddable EJB application.

   Reference the `glassfish-embedded-static-shell.jar` file from the as-install`/lib/embedded` directory of a Eclipse GlassFish installation. Do not move this file or it will not work.

   See Setting the Class Path 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 "Resources and Services Administration" in Eclipse GlassFish Administration Guide. The `jdbc/__default` Java DB database is preconfigured with all distributions of Eclipse GlassFish. However, if you are using `glassfish-embedded-static-shell.jar`, you must start the database manually.

   If your embeddable EJB application uses Java Persistence, you do not need to specify a JDBC resource. See Default Java Persistence Data Source for Embedded Eclipse GlassFish.

3. Invoke one of the `createEJBContainer` methods.

   

   Do not deploy your embeddable EJB application or any of its dependent Jakarta 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` if a nonembedded installation is referenced. This system property applies only to embeddable EJB applications used with nonembedded Eclipse GlassFish.

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

EJB 3.1 Embeddable API Properties

Properties that can be passed to the `EJBContainer#createEJBContainer(Properties)` method are summarized in the following table. All properties are in the `org.glassfish.ejb.embedded.glassfish` package. For example, the full name of the `installation.root` property is `org.glassfish.ejb.embedded.glassfish.installation.root`.

Table 1-11 EJB 3.1 Embeddable API Properties

Property	Default	Description
`installation.root`	Eclipse GlassFish installation location from which `glassfish-embedded-static-shell.jar` is referenced	The Installation Root Directory.

Property	Default	Description
`instance.root`	In order of precedence: • `glassfish.embedded.tmpdir` property value specified in `GlassFishProperties` object • `glassfish.embedded.tmpdir` system property value • `java.io.tmpdir` system property value • as-install`/domains/domain1` if a nonembedded installation is referenced	The Instance Root Directory.
`configuration.file`	domain-dir`/config/domain.xml`	The configuration file.
`keep-temporary-files`	`false`	If `true`, keeps temporary files (exploded EAR file and configuration file) created by the embedded EJB container when Embedded Eclipse GlassFish is stopped.
`web.http.port`	None	Enables the web container if set. Needed for testing web services in a WAR file. The value is ignored and can be an empty string.
`instance.reuse`	`false`	If `true`, no changes are made to the existing configuration file, and a temporary server instance is not created for the embedded run. Instead, execution happens against the existing server instance. Do not use this option if the reused server instance could be in use by the running nonembedded Eclipse GlassFish.
`skip-client-modules`	`false`	If `true`, omits modules from the classpath if they are not specified using `EJBContainer.MODULES` and have a manifest file with a `Main-Class` attribute.

Using Maven with the EJB 3.1 Embeddable API and Embedded Eclipse GlassFish

When using Maven with the EJB 3.1 Embeddable API and Embedded Eclipse GlassFish, you cannot use the features of the Maven plug-in. You must start and stop Embedded Eclipse GlassFish manually or programmatically outside of Maven.

Example 1-14 Maven POM File for Using the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish

This example shows a POM file for configuring Maven to use the EJB 3.1 Embeddable API with Embedded Eclipse GlassFish.

```
<!--
Line breaks in the following element are for readability purposes only
-->
<project 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.example</groupId>
    <artifactId>my-ejb-app-example</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <name>EJB Embeddable API Example</name>
    <dependencies>
        <dependency>
            <groupId>org.glassfish.main.extras</groupId>
            <artifactId>glassfish-embedded-all</artifactId>
            <version>${project.version}</version>
        </dependency>
        <!--
            The jakarta.jakartaee-api is stripped of any code and is just used to
compile your
            application. The scope provided in Maven means that it is used for
compiling,
            but is also available when testing. For this reason, the jakartaee-api
needs to
            be below the embedded GlassFish dependency. The jakartaee-api can actually
be
            omitted when the embedded GlassFish dependency is included, but to keep
your
            project Jakarta EE 10 rather than GlassFish specific, specification is
important.
        -->
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>10.0.0</version>
```

```
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

If you are using `glassfish-embedded-all.jar`, you can omit the `dependency` element with the `jakarta.jakartaee-api` `artifactId`.

Then run `mvn clean verify` command.

Changing Log Levels in Embedded Eclipse GlassFish

To change log levels in Embedded Eclipse GlassFish, you can follow the steps in this section or you can use the Embedded Server API as shown in Example 1-11. For more information about Eclipse GlassFish logging, see "Administering the Logging Service" in Eclipse GlassFish Administration Guide.

You can change log levels in Embedded Eclipse GlassFish in either of the following ways:

- Using the Eclipse GlassFish Embedded Server API
- Creating a custom logging configuration file

Both these ways use logger names. For a list of logger names, use the `list-log-levels` subcommand.

Example 1-15 Using the Eclipse GlassFish Embedded Server API

This example shows how to set log levels using the `getLogger` method in the API.

```
import org.glassfish.embeddable.*;

// Create Embedded GlassFish
GlassFish glassfish = GlassFishRuntime.bootstrap().newGlassFish();

// Set the log levels. For example, set 'deployment' and 'server' log levels to FINEST
Logger.getLogger("").getHandlers()[0].setLevel(Level.FINEST);
Logger.getLogger("jakarta.enterprise.system.tools.deployment").setLevel(Level.FINEST);
Logger.getLogger("jakarta.enterprise.system").setLevel(Level.FINEST);

// Start Embedded GlassFish and deploy an application.
// You will see all the FINEST logs printed on the console.
glassfish.start();
glassfish.getDeployer().deploy(new File("sample.war"));

// Dispose Embedded GlassFish
glassfish.dispose();
```

Example 1-16 Creating a Custom Logging Configuration File

This example shows the contents of a custom logging configuration file, `customlogging.properties`.

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINEST
jakarta.enterprise.system.tools.deployment.level = FINEST
jakarta.enterprise.system.level = FINEST
```

Pass the name of this custom logging configuration file to the `java` command when you invoke

Embedded Eclipse GlassFish. For example:

```
java -Djava.util.logging.config.file=customlogging.properties MyEmbeddedGlassFish
```

Monitoring Embedded Eclipse GlassFish with JMX

Embedded Eclipse GlassFish supports monitoring through JMX MBeans, similar to regular Eclipse GlassFish Server. However, you must attache the flashlight agent on command line to enable monitoring functionality.

AMX (Application Management Extensions) monitoring is only available in `glassfish-embedded-all.jar`. The `glassfish-embedded-web.jar` does not include AMX support, which limits the available monitoring MBeans.

Prerequisites

To enable monitoring in Embedded Eclipse GlassFish, you need the flashlight agent JAR file:

- Download from Maven Central: `org.glassfish.main.flashlight:flashlight-agent`
- Or use from an existing Eclipse GlassFish installation: `glassfish/lib/monitor/flashlight-agent.jar`

Enabling Monitoring Modules

By default, the monitoring service and MBeans support are enabled, but no monitoring modules are active. You can enable specific monitoring modules using one of two methods:

Method 1: Command Line

Start Embedded Eclipse GlassFish with the `enable-monitoring` command:

```
java -javaagent:/path/to/flashlight-agent.jar -jar glassfish-embedded-all.jar 'enable-monitoring --modules thread-pool:http-service'
```

Method 2: Properties File

Create a `glassfish.properties` file in the current directory with monitoring configuration:

```
configs.config.server-config.monitoring-service.module-monitoring-levels.thread-pool=HIGH
configs.config.server-config.monitoring-service.module-monitoring-levels.http-service=HIGH
```

Then start Embedded Eclipse GlassFish:

```
java -javaagent:/path/to/flashlight-agent.jar -jar glassfish-embedded-all.jar
```

Accessing Monitoring Data

Once monitoring modules are enabled, you can access monitoring data through JMX clients such as VisualVM or JConsole.

Using VisualVM

1. Connect to the Embedded Eclipse GlassFish process in VisualVM
2. Install the MBeans extension if not already available
3. Navigate to the MBeans tab
4. Execute the `bootAMX` operation first (this is required, same as with regular Eclipse GlassFish Server)
5. Monitoring data will be available under the `amx` node, in nodes that end with `-mon`, such as `thread-pool-mon` or `request-mon`

Exposing JMX on a Specific Port

To allow remote JMX connections, you can expose the JMX server on a specific port by adding JVM system properties:

```
java -javaagent:/path/to/flashlight-agent.jar \
     -Dcom.sun.management.jmxremote \
     -Dcom.sun.management.jmxremote.port=8686 \
     -Dcom.sun.management.jmxremote.authenticate=false \
     -Dcom.sun.management.jmxremote.ssl=false \
     -jar glassfish-embedded-all.jar 'enable-monitoring --modules thread-pool:http-
service'
```

Exposing JMX without authentication and SSL is not secure and should only be used in development environments. JMX MBeans include management beans that allow executing administrative operations on Eclipse GlassFish, which could be exploited by unauthorized users. In production environments, always enable authentication and SSL for JMX connections.

Available Monitoring Modules

Embedded Eclipse GlassFish supports the same monitoring modules as regular Eclipse GlassFish Server, including:

- `thread-pool` - Thread pool statistics
- `http-service` - HTTP service metrics
- `web-container` - Web container statistics
- `ejb-container` - EJB container metrics
- `transaction-service` - Transaction service data

- `jvm` - JVM statistics

For comprehensive information about available monitoring modules and their metrics, see Administering the Monitoring Service in the Eclipse GlassFish Administration Guide.

Default Java Persistence Data Source for Embedded Eclipse GlassFish

The `jdbc/__default` Java DB database is preconfigured with Embedded Eclipse GlassFish. It is used when an application is deployed in Embedded Eclipse GlassFish that uses Java Persistence but doesn't specify a data source. Embedded Eclipse GlassFish uses the embedded Java DB database created in a temporary domain that is destroyed when Embedded Eclipse GlassFish is stopped. You can use a Java DB database configured with nonembedded Eclipse GlassFish if you explicitly specify the instance root directory or the configuration file.

By default, weaving is enabled when the Eclipse GlassFish Embedded Server API is used. To disable weaving, set the `org.glassfish.persistence.embedded.weaving.enabled` property to `false`.

Restrictions for Embedded Eclipse GlassFish

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

- The following web technologies of the Jakarta EE platform:
  - Jakarta Servlet API
  - Jakarta Pages
  - Jakarta Faces
- JDBC connection pooling
- Jakarta Persistence API
- Jakarta Transaction API
- Jakarta Transaction Service

The `glassfish-embedded-all.jar` and `glassfish-embedded-static-shell.jar` files support all features of nonembedded Eclipse GlassFish with these exceptions:

- Installers
- Administration Console
- Apache Felix OSGi framework
- The Maven plug-in for embedded Eclipse GlassFish 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 Eclipse GlassFish if a nonembedded Eclipse GlassFish is running in parallel.

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

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

When embedded Eclipse GlassFish is started, the base installation directory that Eclipse GlassFish uses depends on the options with which Eclipse GlassFish is started. If necessary, embedded Eclipse GlassFish creates a base installation directory. Embedded Eclipse GlassFish then copies the following directories and their contents from the Java archive (JAR) file in which embedded Eclipse GlassFish is distributed:

- `domains`
- `lib`

If necessary, Eclipse GlassFish also creates an instance root directory.