Eclipse GlassFish Server Embedded Server Guide, Release 5.1

Table of Contents

Eclipse GlassFish Server

Embedded Server Guide

Release 5.1

Contributed 2018, 2019

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

Note: The main thrust of the Eclipse GlassFish Server 5.1 release is to provide an application server for developers to explore and begin exploiting the new and updated technologies in the Java EE 7 platform. Thus, the embedded server feature of GlassFish Server 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 Java EE 7 platform.

Eclipse GlassFish Server Embedded Server Guide, Release 5.1

Copyright ?? 2013, 2019 Oracle and/or its affiliates. All rights reserved.

This program and the accompanying materials are made available under the terms of the Eclipse Public License v. 2.0, which is available at http://www.eclipse.org/legal/epl-2.0.

SPDX-License-Identifier: EPL-2.0

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

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 Java 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 GlassFish Server Open Source Edition and to develop applications in which GlassFish Server is embedded. This document is for software developers who are developing applications to run in embedded GlassFish Server. The ability to program in the Java language is assumed.

Note:

The main thrust of the GlassFish Server Open Source Edition 4.0 release is to provide an application server for developers to explore and begin exploiting the new and updated technologies in the Java EE 7 platform. Thus, the embedded server feature of GlassFish Server 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 Java EE 7 platform.

This preface contains information about and conventions for the entire GlassFish Server Open Source Edition (GlassFish Server) documentation set.

GlassFish Server 4.0 is developed through the GlassFish project open-source community at `http://glassfish.java.net/`. The GlassFish project provides a structured process for developing the GlassFish 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 GlassFish Server source code and to contribute to the development of the GlassFish Server. The GlassFish project is designed to encourage communication between Oracle engineers and the community.

GlassFish Server Documentation Set

The GlassFish Server documentation set describes deployment planning and system installation. For an introduction to GlassFish Server, 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.
Quick Start Guide	Explains how to get started with the GlassFish Server product.
Installation Guide	Explains how to install the software and its components.
Upgrade Guide	Explains how to upgrade to the latest version of GlassFish Server. 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 GlassFish Server that meets the requirements of your system and enterprise.
Administration Guide	Explains how to configure, monitor, and manage GlassFish Server subsystems and components from the command line by using the `asadmin` 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 GlassFish Server security.
Application Deployment Guide	Explains how to assemble and deploy applications to the GlassFish Server and provides information about deployment descriptors.
Application Development Guide	Explains how to create and implement Java Platform, Enterprise Edition (Java EE platform) applications that are intended to run on the GlassFish Server. These applications follow the open Java standards model for Java EE components and application programmer interfaces (APIs). This guide provides information about developer tools, security, and debugging.
Embedded Server Guide	Explains how to run applications in embedded GlassFish Server and to develop applications in which GlassFish Server is embedded.
High Availability Administration Guide	Explains how to configure GlassFish Server to provide higher availability and scalability through failover and load balancing.
Performance Tuning Guide	Explains how to optimize the performance of GlassFish Server.
Troubleshooting Guide	Describes common problems that you might encounter when using GlassFish Server and explains how to solve them.

Book Title	Description
Error Message Reference	Describes error messages that you might encounter when using GlassFish Server.
Reference Manual	Provides reference information in man page format for GlassFish Server administration commands, utility commands, and related concepts.
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 GlassFish Server.
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 Java EE applications:

- Your First Cup: An Introduction to the Java EE Platform (`http://docs.oracle.com/javaee/7/firstcup/doc/home.html`). For beginning Java 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 Java EE 7 Tutorial (`http://docs.oracle.com/javaee/7/tutorial/doc/home.html`). This comprehensive tutorial explains how to use Java EE 7 platform technologies and APIs to develop Java EE applications.

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

- The API specification for version 7 of Java EE is located at `http://docs.oracle.com/javaee/7/api/`.

- The API specification for GlassFish Server 4.0, including Java EE 7 platform packages and nonplatform packages that are specific to the GlassFish Server product, is located at `http://glassfish.java.net/nonav/docs/v3/api/`.

Additionally, the Java EE Specifications (`http://www.oracle.com/technetwork/java/javaee/tech/index.html`) might be useful.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see the NetBeans Documentation, Training & Support page (`http://www.netbeans.org/kb/`).

For information about the Java DB database for use with the GlassFish Server, see the Java DB product page (`http://www.oracle.com/technetwork/java/javadb/overview/index.html`).

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

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.
>	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 GlassFish Server. 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`/glassfish3/glassfish` Installations on the Windows operating system: SystemDrive`:\glassfish3\glassfish`

Placeholder	Description	Default Value
as-install-parent	Represents the parent of the base installation directory for GlassFish Server.	Installations on the Oracle Solaris operating system, Linux operating system, and Mac operating system: user's-home-directory`/glassfish3` Installations on the Windows operating system: SystemDrive`:\glassfish3`
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

1 Eclipse GlassFish Server 5.1 Embedded Server Guide

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

The following topics are addressed here:

Introduction to Embedded GlassFish Server

Embedded Eclipse GlassFish Server enables you to use GlassFish Server as a library. Embedded GlassFish Server also enables you to run GlassFish Server inside any Virtual Machine for the Java platform (Java Virtual Machine or JVMmachine).

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

> Note:
>
> Embedded GlassFish Server does not run on the Java Platform, Micro Edition (Java ME platform).

You can use embedded GlassFish Server in the following ways:

- With the Embedded Server API (see Including the GlassFish Server Embedded Server API in Applications)
- With the Maven Plug-in (see Testing Applications with the Maven Plug-in for Embedded GlassFish

Server)

- With the EJB 3.1 Embeddable API (see Using the EJB 3.1 Embeddable API with Embedded GlassFish Server)

Embedded GlassFish Server 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 GlassFish Server on a single system. Testing and debugging are simplified because the need for an installed and configured GlassFish Server is eliminated. Therefore, you can run unit tests automatically in every build.

Note:

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

Embedded GlassFish Server File System

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

- 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 GlassFish Server uses for configuration files. This directory corresponds to the base directory for an installation of GlassFish Server. Configuration files are contained in the following directories in the base directory for an installation of GlassFish Server:

- `domains`
- `lib`

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

Instance Root Directory

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

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

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

If domain-dir is not specified, GlassFish Server creates a directory named `gfembed`random-number`tmp` in a temporary directory, where random-number is a randomly generated 19-digit number. GlassFish Server 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 GlassFish 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 GlassFish Server. To specify an existing `domain.xml` file, invoke the `setConfigFileURI` method of the `GlassFishProperties` class as described in Creating and Configuring an Embedded GlassFish Server.

> Note:
>
> The built-in `domain.xml` file used by default by Embedded GlassFish Server can be downloaded from `http://embedded-glassfish.java.net/domain.xml`. You can customize this file and pass it in using the `setConfigFileURI` method while creating an Embedded GlassFish Server.

Including the GlassFish Server Embedded Server API in Applications

Eclipse GlassFish Server provides an application programming interface (API) for developing applications in which GlassFish Server is embedded. For details, see the `org.glassfish.embeddable` packages at http://embedded-glassfish.java.net/nonav/apidocs/.

The following topics are addressed here:

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

Setting the Class Path

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

`glassfish-embedded-nucleus.jar`
: Corresponds to the nucleus distribution. Download this file from http://download.java.net/maven/glassfish/org/glassfish/extras/glassfish-embedded-nucleus/.

`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-web/.

`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-all/.

`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 GlassFish Server. Reference this file from the as-install`/lib/embedded` directory of a nonembedded GlassFish Server installation. Do not move this file or it will not work. For an explanation of as-install, see Installation Root Directory.

Note:

Oracle GlassFish Server only supports use of the `glassfish-embedded-static-shell.jar` file. The other files are part of Eclipse GlassFish Server and are offered without official support.

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 GlassFish Server

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

The following topics are addressed here:

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

Creating and Configuring an Embedded GlassFish Server

To create and configure an embedded GlassFish Server, 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.tmp` 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`
Sets the port on which Embedded GlassFish Server listens.	`setPort`(String networkListener, int port)	none

Note:

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

Example 1-1 Creating an Embedded GlassFish Server

This example shows code for creating an Embedded GlassFish Server.

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

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

This example shows code for creating an Embedded GlassFish Server 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\\domai
n1");
    GlassFish glassfish = glassfishRuntime.newGlassFish(glassfishProperties);

    glassfish.start();

...
```

Running an Embedded GlassFish Server

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

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

Setting the Port of an Embedded GlassFish Server 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.

> Note:
>
> 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 GlassFish Server

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

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

Starting an Embedded GlassFish Server From an Application

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

Example 1-4 Starting an Embedded GlassFish Server

This example shows code for setting the port and starting an embedded GlassFish Server. 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 GlassFish Server From an Application

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

Example 1-5 Stopping an Embedded GlassFish Server

This example shows code for prompting the user to press the Enter key to stop an embedded GlassFish Server. 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 Server
...
```

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

Deploying and Undeploying an Application in an Embedded GlassFish Server

Deploying an application installs the files that comprise the application into Embedded GlassFish Server 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 GlassFish Server, see the GlassFish Server Open Source Edition 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 "Deploying Applications" in GlassFish Server Open Source Edition 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 GlassFish 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 GlassFish Server Open Source Edition 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`,` 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 GlassFish Server. For example, before stopping GlassFish Server, undeploy all applications that are running in GlassFish

Server.

> Note:
>
> If you reference a nonembedded GlassFish Server 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 GlassFish Server Embedded Server API ^^^

```
...
import org.glassfish.embeddable.*;
...
    String command = "set-log-level";
    String weblevel = "javax.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 GlassFish Server.

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 GlassFish Server

If you are using Apache Maven (`http://maven.apache.org/`), the plug-in for embedded GlassFish Server 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 GlassFish Server
- To Redeploy an Application That Was Built and Started From Maven
- Maven Goals for Embedded GlassFish Server

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

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

To Set Up Your Maven Environment

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

1. Identify the Maven plug-in for embedded GlassFish Server.
   Add the following `plugin` element to your POM file:

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

version

The version to use. The version of the final promoted build for this release is `3.1`. The Maven plug-in is not bound to a specific version of GlassFish Server. You can specify the version you want to use. If no version is specified, a default version, 3.1 for this release, is used.

1. 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>target/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 GlassFish Server. 3. Perform advanced plug-in configuration. This step is optional.

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

```
...
        <plugins>
            ...
            <plugin>
                ...
                <configuration>
                    <goalPrefix>embedded-glassfish</goalPrefix>
                    <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>embedded-glassfish-config.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>
...
```

1. 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 GlassFish Server.

1. 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 1-13 POM File for Configuring Maven to Use Embedded GlassFish Server

This example shows a POM file for configuring Maven to use embedded GlassFish 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>3.1</version>
  <name>Maven test</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.glassfish</groupId>
        <artifactId>maven-embedded-glassfish-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <goalPrefix>embedded-glassfish</goalPrefix>
          <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>
  <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 GlassFish Server.

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 GlassFish Server. * Deploys your application.
The application continues to run in Embedded GlassFish Server until Embedded GlassFish Server is stopped.

To Stop Embedded GlassFish 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 GlassFish Server.

```
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 GlassFish Server 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 GlassFish Server until Embedded GlassFish Server 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 GlassFish Server

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

- `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.tmp` 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 Server 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 GlassFish Server 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

Parameter	Default	Description
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.
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 GlassFish 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 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 Server Application Deployment Guide.	(optional) The name of the application.
contextRoot	The name of the application.	(optional) The context root of the application.

Parameter	Default	Description
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.
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

Note:

If you reference a nonembedded GlassFish Server 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.

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 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 GlassFish Server 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 GlassFish Server Open Source Edition 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 `http://glassfish.java.net/nonav/docs/v3/api/`.

Parameter	Default	Description
commandLine	None	The full `asadmin` syntax of the command.

Using the EJB 3.1 Embeddable API with Embedded GlassFish 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 GlassFish Server instance. However, you can take advantage of Embedded GlassFish Server's ease of use by referencing the nonembedded GlassFish Server instance with the `glassfish-embedded-static-shell.jar` file.

Embedded GlassFish Server 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 GlassFish Server.

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 7 Downloads (`http://www.oracle.com/technetwork/java/javaee/downloads/index.html`) or Code Samples (`http://www.oracle.com/technetwork/java/javaee/documentation/index.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 GlassFish Server
- EJB 3.1 Embeddable API Properties
- Using Maven with the EJB 3.1 Embeddable API and Embedded GlassFish Server

To Use the EJB 3.1 Embeddable API with Embedded GlassFish Server

1. To specify GlassFish Server 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 GlassFish Server 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 Server Administration Guide. The `jdbc/__default` Java DB database is preconfigured with all distributions of GlassFish Server. 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 GlassFish 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.

1. 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 GlassFish Server.
2. 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`	GlassFish Server 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.tmp` 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 GlassFish Server 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 GlassFish Server.
`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 GlassFish Server ^^^^^^^^^^^^^^^^^^^^^^^^^

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

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

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

```
<!--
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.glassfish</groupId>
    <artifactId>my-ejb-app-tester</artifactId>
    <version>3.1</version>
    <name>Maven test</name>
    <dependencies>
        <dependency>
            <groupId>org.glassfish.extras</groupId>
            <artifactId>glassfish-embedded-static-shell</artifactId>
            <version>${project.version}</version>
            <scope>system</scope>
            <systemPath>
                ${env.S1AS_HOME}/lib/embedded/glassfish-embedded-static-shell.jar
            </systemPath>
        </dependency>
<!--
        The javaee-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 javaee-api needs to
        be below the embedded Glassfish dependency. The javaee-api can actually be
        omitted when the embedded Glassfish dependency is included, but to keep your
        project Java-EE 6 rather than GlassFish 3, specification is important.
-->
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>6.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <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>
```

If you are using `glassfish-embedded-static-shell.jar`, you can omit the `dependency` element with the `javaee-api` `artifactId` and the `pluginRepositories` element.

Set the `S1AS_HOME` environment variable to the installation root directory before running the `mvn clean verify` command.

Changing Log Levels in Embedded GlassFish Server

To change log levels in Embedded GlassFish Server, 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 GlassFish Server logging, see "Administering the Logging Service" in Eclipse GlassFish Server Administration Guide.

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

- Using the GlassFish Server 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 GlassFish Server 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("javax.enterprise.system.tools.deployment").setLevel(Level.FINEST);
Logger.getLogger("javax.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
javax.enterprise.system.tools.deployment.level = FINEST
javax.enterprise.system.level = FINEST
```

Pass the name of this custom logging configuration file to the `java` command when you invoke Embedded GlassFish Server. For example:

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

Default Java Persistence Data Source for Embedded GlassFish Server

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

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

Restrictions for Embedded GlassFish Server

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

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

- Java Transaction API
- Java Transaction Service

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

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

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

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

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

- `domains`
- `lib`

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