Sun GlassFish Enterprise Server v3 Scripting Framework Guide

Part No: 820–7697–11
December 2009

100608@20205

Contents

Preface

Sun GlassFish Enterprise Server v3 Scripting Framework Guide explains how to develop scripting applications in languages such as Ruby on Rails and Groovy on Grails for deployment to Enterprise Server.

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

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

The following topics are addressed here:

- "Enterprise Server Documentation Set" on page 7
- "Related Documentation" on page 9
- "Typographic Conventions" on page 10
- "Symbol Conventions" on page 10
- "Default Paths and File Names" on page 11
- "Documentation, Support, and Training" on page 12
- "Searching Sun Product Documentation" on page 12
- "Third-Party Web Site References" on page 12
- "Sun Welcomes Your Comments" on page 12

Enterprise Server Documentation Set

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

TABLE P–1 Books in the Enterprise Server Documentation Set

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

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

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

Related Documentation

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

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

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

Additionally, the following resources might be useful:

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

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

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

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

Typographic Conventions

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

TABLE P–2 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–3 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.

TABLE P–3 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Default Paths and File Names

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

TABLE P–4 Default Paths and File Names

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

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 820-7967.

◆ ◆ ◆ CHAPTER 1

Using JRuby on Rails With Sun GlassFish Enterprise Server

This tutorial shows you how to get started using JRuby on Rails on the Sun GlassFish Enterprise Server v3.

The following topics are addressed here:

- "Introduction to JRuby and Rails on Sun GlassFish Enterprise Server" on page 13
- "Installation and Configuration of JRuby" on page 15
- "Enterprise Server v3 JRuby Container Configuration" on page 17
- "Creating a Simple Rails Application" on page 19
- "Deploying and Running a Rails Application" on page 22
- "Accessing a Database From a Rails Application" on page 23
- "Accessing Java Libraries From a Rails Application" on page 24
- "Monitor Rails Applications on Enterprise Server v3" on page 28
- "GlassFish v3 Gem" on page 29
- "Introduction to Warbler" on page 31
- "Creating and Deploying a Simple Rails Application with Warbler" on page 32
- "Further Information" on page 33

Introduction to JRuby and Rails on Sun GlassFish Enterprise Server

This section gives you an overview of JRuby and Rails on the Sun GlassFish Enterprise Server by explaining the following concepts:

- "What Is Ruby on Rails ?" on page 14
- "What Is JRuby ?" on page 14
- "JRuby on Rails, the Sun GlassFish Enterprise Server v3, and the GlassFish v3 Gem" on page 14

What Is Ruby on Rails ?

Ruby is an interpreted, dynamically-typed, object-oriented programming language. It has a simple, natural syntax that enables developers to create applications quickly and easily. It also includes the easy-to-use RubyGems packaging utility for customizing a Ruby installation with additional plug-ins.

Rails is a web application framework that leverages the simplicity of Ruby and eliminates much of the repetition and configuration required in other programming environments. With Rails, you can create database-backed web applications, complete with models and tables, by running a few one-line commands.

To learn more about Ruby on Rails, see Ruby on Rails.

What Is JRuby ?

JRuby is a Java implementation of the Ruby interpreter. While retaining many of the popular characteristics of Ruby, such as dynamic-typing, JRuby is integrated with the Java platform. With JRuby on Rails, you get the simplicity and productivity offered by Ruby and Rails and the power of the Java platform offered by JRuby, thereby giving you many benefits as a Rails developer, including the following:

- You can access the rich set of Java libraries from your Rails application.
- You can use the powerful and secure support of Java Unicode strings with your Rails application.
- Your JRuby on Rails application can spin off multiple threads because JRuby uses Java threads, which map to native Ruby threads. Furthermore, you can pool these threads.

To learn more about JRuby, see JRuby.

JRuby on Rails, the Sun GlassFish Enterprise Server v3, and the GlassFish v3 Gem

Developing and deploying your Rails application on the Sun GlassFish Enterprise Server gives you the following advantages over using a typical web server used for running Rails applications:

- A simple, integrated deployment environment. In other words, you do not need one set of software for developing the application and another set of software for deploying it.
- The ability to deploy multiple Rails applications to a single GlassFish Server instance.
- The ability of a Rails application to handle multiple requests.

You have the following options for deploying a Rails application on GlassFish Server:

- Deploy the application as a directory to the Enterprise Server v3.
- Run the application using GlassFish v3 Gem.

A Gem is a Ruby package that contains a library or an application. In fact, Rails itself is a Gem that is installed on JRuby. For more details on GlassFish v3 Gem, see "GlassFish v3 Gem" on page 29.

Installation and Configuration of JRuby

To develop and deploy Rails applications on the Enterprise Server, do the following:

1. Download and install JRuby.
2. Install Rails on top of your JRuby installation.

You can perform the above tasks by installing JRuby on your Enterprise Server instance in one of the following ways:

- Installing JRuby and the associated Gems on your Enterprise Server from Update Tool.
- Installing JRuby and Rails Gem directly

▼ To Install JRuby and Rails from Update Center

JRuby and other associated Gems are now available as IPS packages from Update Center. By downloading them using the Update Tool, you can install them directly on your Enterprise Server.

For information about the Update Tool, see *Sun GlassFish Enterprise Server v3 Installation Guide*.

1 Start the update tool:

as-install`/bin/updatetool`

2 From the Update Tool, choose the following packages from Available Add-Ons:

- `JRuby on GlassFish` which contains JRuby 1.3.1
- `JRuby Gems` which contains Rails 2.3.2, Warbler, `jdbc-mysql`, and `activerecord-jdbcmysql-adapter` packages.

3 Click Install, which will install the packages on your Enterprise Server instance.

4 Set your `JRUBY_HOME` environment variable to the location of your JRuby installation.

`export JRUBY_HOME=/`*jruby-install-location*

5 **Add `JRUBY_HOME/bin` directory to your system path so that you can invoke JRuby from anywhere in your directory tree.**

```
export PATH=$PATH:$JRUBY_HOME/bin
```

6 **On the Windows operating system, use the following commands for Steps 4 and 5:**

set JRUBY_HOME=C:\jruby-install-location

```
set PATH=%JRUBY_HOME%\bin;%PATH%
```

Note – If GlassFish v3 JRuby IPS package was installed using update tool, then there is no need to set the `jruby.home` system property

---

▼ To Install JRuby as Standalone

To install your own JRuby instance as standalone, use the following procedure.

1 **Go to JRuby download site JRuby Download Site.**

2 **Download `jruby-bin-1.3.1.zip` or the latest version.**

3 **Unpack the zip file:**

```
unzip jruby-bin-1.3.1.zip
```

4 **Set your `JRUBY_HOME` environment variable to the location of your JRuby installation:**

export JRUBY_HOME=/*jruby-install-location*

5 **Add `JRUBY_HOME/bin` directory to your system path so that you can invoke JRuby from anywhere in your directory tree:**

```
export PATH=$PATH:$JRUBY_HOME/bin
```

6 **If you want to use this JRuby installation with your Enterprise Server, use the following steps to inform and configure Enterprise Server instance with the location of the JRuby installation:**

a. **Start your GlassFish installation:**

```
asadmin start-domain
```

b. **Set JRuby home:**

asadmin configure-jruby-container --jruby.home=/*jruby-install-location*

▼ To Install Rails Gem on JRuby

If you installed your JRuby as a standalone instance, you also need to install the required Rails and other Gems on it. To install Rails, use the following procedure.

● **Install the Rails Gem:**

```
jruby –S gem install rails
```

The `-S` parameter that you used to run the command to install Rails tells JRuby to look for the script anywhere in the *JRUBY_HOME* path. With this command, JRuby and required Gems are installed on JRuby.

Enterprise Server v3 JRuby Container Configuration

The Sun GlassFish Enterprise Server v3 provides the following ways to configure the installed JRuby container.

- Asadmin CLI
- Administration Console

Configuring JRuby Container Through Asadmin CLI

The `asadmin` CLI now provides options to configure the JRuby container. The command execution is reflected in changes to the JRuby container configuration section of the `domain.xml` file which makes them persistent.

The following JRuby container properties can be configured through the `asdmin` command.

```
configure-jruby-container [--help]
     [--monitoring={false|true}]
     [--jruby-home jruby-home]
     [--jruby-runtime jruby-runtime]
     [--jruby-runtime-min jruby-runtime-min]
     [--jruby-runtime-max jruby-runtime-max]
     [--show={true|false}]
```

Use the following `asadmin` command syntax to configure these values:

```
asadmin configure-jruby-container --<property>=<value>
```

For example, the following command is used to set the JRuby home if JRuby instance is installed as standalone:

```
asadmin configure-jruby-container --jruby.home=/<jruby-install-location>
```

You can also change the deployment specific options to the JRuby application through the following command syntax:

```
asadmin deploy --property <name>=<value>[:<name>=<value>]
```

For example, the following command uses a JRuby instance to deploy your application which is different from the one configured with the Enterprise Server:

```
asadmin deploy --property jruby.home=/<latest-jruby-install> <application>
```

For a detailed description of these options, see `configure-jruby-container(1)`.

The JRuby container runtime pool options are discussed in the next section.

Configuring JRuby Runtime Pool

The Sun GlassFish Enterprise Server v3 provides a JRuby runtime pool to allow servicing of multiple concurrent requests. However Rails is not currently thread-safe, and while JRuby is able to take advantage of Java's native threading, Rails cannot benefit from it. Each JRuby runtime runs a single instance of Rails, and requests are handed off to whichever instance happens to be available at the time of the request.

The JRuby properties in the above configuration are explained as follows:

- *jruby.runtime* property sets the initial number of JRuby runtimes that Enterprise Server starts with. The default value is one. This represents the highest value that GlassFish Server accepts as minimum runtimes, and the lowest value that GlassFish can use as maximum runtimes.
- *jruby.runtime.min* property sets the minimum number of JRuby runtimes that will be available in the pool. The default value is one. The pool will always be at least this large, but can be larger than this.
- *jruby.runtime.max* property sets the maximum number of JRuby runtimes that might be available in the pool. The default value is two. Setting Too high values for this property might result in `OutOfMemory` errors, either in the heap or in the `PermGen`.

The dynamic runtime pool maintains itself with the minimum number of runtimes possible, to allow consistent and fast runtime access for the requesting application. The pool may take a initial runtime value, but that value is not used after pool creation.

The JRuby runtime pool values can be set either at container level or at deploy time. To set the runtime pool values at container level, use the following `asadmin` command:

```
asadmin configure-jruby-container --jruby-runtime=2 --jruby-runtime-min=1
--jruby-runtime-max=3
```

Note that each of the above properties can be set separately at a time. To set the runtime pool values at deploy time use the following `asadmin` command:

```
asadmin deploy --property
jruby-runtime=2:jruby-runtime-min=1:jruby-runtime-max=3
```

Each of the above properties also can be set at a time or at the same time. If both settings are used, deploy time settings take precedence over container runtime settings.

Configuring JRuby Container Through Administration Console

The JRuby container values can be set from Enterprise Server Administration Console. For details on Administration Console, see "Administration Console" in *Sun GlassFish Enterprise Server v3 Administration Guide*.

You can set the following container properties from the Administration Console:

- JRuby Home
- Initial Pool Size
- Minimum Pool Size
- Maximum Pool Size

Use the following steps to access JRuby Container from Administration Console.

▼ To Configure JRuby Container from Administration Console

1 **Access the Enterprise Server Administration Console from web browser. For example:**

```
http://localhost:4848
```

2 **Login to the Administration Consoleif configured for secure login.**

3 **Select Common Tasks→Configuration→JRuby Container on the left-hand tree panel.**

4 **Change JRuby Container values from the details panel.**

Creating a Simple Rails Application

After completing your installations, you are ready to start coding. This section shows you how to create a simple application that displays the following message:

```
Welcome to JRuby on Rails on the Sun GlassFish Enterprise Server!
```

▼ To Create the `hello` Application

1 **Select a directory to create a sample application. For example:**

```
/apps/jruby-apps
```

2 **Create a Rails application called `hello` in that directory:**

```
cd /apps/jruby-apps
```

```
jruby -S rails hello
```

This command creates the `hello` directory, which contains a set of automatically-generated files and directories.

The directories containing the files that you'll use the most are:

- `app`: Contains your application code.
- `config`: Contains configuration files, such as `database.yml`, which you use to configure a database.
- `public`: Contains files and resources that need to be accessed directly rather than accessed through the Rails call stack. These include images and straight HTML files.

▼ To Create the Controller and the View

The following task describes how to create a controller and a default view for your application. The controller handles requests, dispatches them to other parts of the application as necessary, and determines which view to render. The view is the file that generates the output to the browser. In Rails, views are typically written with ErB, a templating mechanism.

1 **Change to the directory where you created the `hello` application in the previous task. For example:**

```
/apps/jruby-apps/hello
```

2 **Create a controller and default view for your application:**

```
jruby script/generate controller home index
```

You should see a controller file called `home_controller.rb` in the `hello/app/controllers` directory and a view file called `index.html.erb` in the `hello/app/views` directory.

▼ To Pass Data From the Controller to the View

Exchanging data between the controller and the views is a common task in web application development. This section describes how to set an instance variable in the controller and access its value from the view.

1 **Go to the controller that you created in the previous task.**

2 **Open the `hello/app/controllers/home_controller.rb` file in a text editor.**

3 **Add an instance variable called `@hello_message` to the action called `index`, so that the controller looks like this:**

```
class HomeController < ApplicationController
def index
@hello_message = "Welcome to JRuby on Rails on the Sun GlassFish Enterprise Server"
end
end
```

In Rails, the actions are supposed to map to views. So, when you access the `index.html.erb` file, the index action executes. In this case, the index action makes the `@hello_message` variable available to `index.html.erb`.

4 **Save the file.**

5 **Open the `hello/app/views/home/index.html.erb` file in a text editor.**

6 **At the end of the file, add the following output block:**

```
<%= @hello_message %>
```

This JRuby code, embedded into the view, inserts the value of `@hello_message` into the page. When you run the application, you can see "Welcome to JRuby on Rails on the GlassFish Enterprise Server" in your browser.

7 **Save the file.**

▼ To Use Rails Without a Database

Although Rails is intended for creating database-backed web applications, this example is simple enough that it doesn't require one. In this case, you must edit the `enviroment.rb` configuration file to indicate that your application does not use a database.

1 **Open `hello/config/environment.rb` file in a text editor.**

2 **Look for the following pattern in the file:**

```
#config.frameworks -= [ :active_record, :active_resource, :action_mailer ]
```

3 **Remove the pound (#) character in the beginning of the line. The line should now read as follows:**

```
config.frameworks -= [ :active_record, :active_resource, :action_mailer ]
```

4 **Exit and save the file.**

Deploying and Running a Rails Application

As described in "JRuby on Rails, the Sun GlassFish Enterprise Server v3, and the GlassFish v3 Gem" on page 14, you have two ways to deploy your Rails application on the Enterprise Server:

- Deploy it as a directory to the Enterprise Server.
- Run it using the GlassFish v3 Gem.

For more details on running the Rails application using GlassFish v3 Gem, see "To Run a Rails Application on GlassFish v3 Gem" on page 30. This section shows you how to deploy the `hello` application that you created in the previous section, as a directory with the `asdmin` CLI and how to access the application from your web browser. You can also use these same instructions to deploy a legacy Rails application.

▼ To Deploy the Rails Application as a Directory

You can use directory-based deployment to deploy any Rails application to the Enterprise Server. To deploy the `hello` application to the Enterprise Server as a directory, do the following:

1. **Start the Enterprise Server with the following command:**

   ```
   asadmin start-domain
   ```

2. **Change to the directory where you created the sample application. For example:**

   ```
   /apps/jruby-apps/hello
   ```

3. **Deploy the `hello` application with `asadmin` command:**

   ```
   asadmin deploy hello
   ```

   - **If you want to use a JRuby instance different from the update installation or the instance you configured with `configure-jruby-container` command, you can use the following deploy time option:**

     `asadmin deploy --property jruby.home=`*/jruby-install-location*

4. **Run the `hello` application using the following URL in your browser:**

   ```
   http://localhost:8080/hello/
   ```

Accessing a Database From a Rails Application

One of the main functions of Rails is to make a quick-and-easy task of creating an application that accesses a database. This section shows you the steps to create a simple application that accesses a book database using MySQL. You should have already installed JRuby, Rails, and the required Gems.

▼ To Set Up the MySQL Database Server

1. **Download and install MySQL database server:**

   MySQL 5.0 Community Server.

2. **Configure the server according to the MySQL documentation, including entering a root password.**

3. **Start the server.**

▼ To Create a Database-Backed Rails Application

1. **Create or select a directory for creating a database-backed Rails application.**

2. **Change to that directory.**

3. **Create and configure a application template to use the MySQL database:**

   ```
   jruby -S rails books -d mysql
   ```

4. **Change to the `books` directory that you just created.**

5. **Replace `mysql` with `jdbcmysql`.**

6. **Open the `config/database.yml` file in a text editor.**

7. **When prompted, enter your MySQL root password under the development heading in the `database.yml` file.**

8. **Change back to the `books` directory if you are not already there.**

9. **Create the database by running the following command:**

   ```
   jruby -S rake db:create
   ```

After the database creation is complete, you should see output similar to the following:

```
** Execute db:create
```

The `rake` command invokes the Rake tool. The Rake tool builds applications by running Rake files, which are written in Ruby and provide instructions for building applications.

10 **Create the scaffold and the `Book` model for the application:**

```
jruby script/generate scaffold book title:string
author:string isbn:string description:text
```

When you run the `script/generate` command, you specify the name of the model, the names of the columns, and the types for the data contained in the columns.

A scaffold is the set of code that Rails generates to handle database operations for a model object, which is `Book` in this case. The scaffold consists of a controller and some views that allow users to perform the basic operations on a database, such as viewing the data, adding new records, and editing records. Rails also creates the model object when generating the scaffold.

11 **Create the database tables:**

```
jruby -S rake db:migrate
```

When Rails is finished creating the tables, you should see output similar to the following:

```
CreateBooks: migrated (0.1322ms) =========
```

If you need to reset the database later, you can run the following command:

```
jruby –S rake db:reset
```

Accessing Java Libraries From a Rails Application

The primary advantage of developing with JRuby is that you have access to Java libraries from a Rails application. For example, say you might want to create an image database and a web application that allows processing of the images. You can use Rails to set up the database-backed web application and use the powerful Java 2D API for processing the images on the server-side.

This section shows you how to get started using Java libraries in a Rails application while stepping you through building a simple Rails application that does basic image processing with the Java 2D API.

This application demonstrates the following concepts involved in using Java libraries in a Rails application:

- Giving your controller access to Java libraries
- Creating constants to refer to Java classes
- Performing file input and output using the `java.io` and `javax.imageio` packages
- Assigning Java objects to Ruby objects

- Calling Java methods and using variables
- Converting arrays from Java language arrays to Ruby arrays
- Streaming files to the client

For simplicity's sake, this application does not use a database. You will need a `JPEG` file to run this application.

▼ To Create the Rails Application That Accesses Java Libraries

1. **Change to the directory you want to create an application.**

   ```
   /apps/jruby-apps/
   ```

2. **Create an application by running this command:**

   ```
   jruby -S rails imageprocess
   ```

3. **Open the `imageprocess/config/environment.rb` file in a text editor.**

4. **Follow steps 2 and 3 from the instructions in, "To Use Rails Without a Database" on page 21.**

5. **Change to the `imageprocess` directory you just created.**

6. **Create a controller and default view for the application by running this command:**

   ```
   jruby script/generate controller home index
   ```

7. **Change to the `imageprocess/app/views/home` directory.**

8. **Create a second view by copying the default view to `seeimage.html.erb`:**

   ```
   cp index.html.erb seeimage.html.erb
   ```

▼ To Create the Views That Display the Images Generated by Java2D Code

With this task, you will perform the following actions:

- Load an image on which you want to perform image processing with Java2D.
- Make the initial view show the original image and provide a link that the user clicks to perform the `ColorConvertOp` image processing operation.
- Make the other view display the processed image.

1. **Find a `JPEG` image that you can use with this application.**

2 **Add the image to `imageprocess/public/image` file.**

3 **Change to `imageprocess/app/views/home` directory.**

4 **Open the `index.html.erb` file in a text editor.**

5 **Replace the contents of this file with the following HTML markup:**

```
<html>
    <body>
        <img src="../../images/kids.jpg"/><p>
        <%= link_to "Perform a ColorConvertOp on this image", :action => "seeimage" %>
    </body>
</html>
```

This page loads an image from `imageprocess/public/images` and provides a link that references the `seeimage` action. The `seeimage` action maps to the `seeimage` view, which shows the processed image.

6 **Replace `kids.jpg` from line 3 of `index.html.erb` with the name of your image that you saved from Step 3 of this procedure.**

7 **Save the `index.html.erb` file.**

8 **Open the `seeimage.html.erb` file in a text editor.**

9 **Replace the contents of this file with the following HTML markup:**

```
<html>
    <body>
        <img src="/home/processimage"/><p>
        <%= link_to "Back", :action => "index" %>
    </body>
</html>
```

The `img` tag on this page accesses the `processimage` action in `HomeController`. The `processimage` action is where you will put the Java2D code to process the image that you loaded into `index.html.erb`.

▼ To Add Java2D Code to a Rails Controller

With this task, you will add the code to process your `JPEG` image.

1 **Add the following line to `HomeController`, right after the class declaration:**

```
include Java
```

This line is necessary for you to access any Java libraries from your controller.

2 **Create a constant for the `BufferedImage` class so that you can refer to it by the shorter name:**

```
BI = java.awt.image.BufferedImage
```

3 **Add an empty action, called `seeimage`, at the end of the controller:**

```
def seeimage
end
```

This action is mapped to the `seeimage.html.erb` view.

4 **Give controller access to your image file using `java.io.File`, making sure to use the name of your image in the path to the image file. Place the following line inside the `seeimage` action:**

```
filename = "#{RAILS_ROOT}/public/images/kids.jpg"
imagefile = java.io.File.new(filename)
```

Notice that you don't need to declare the types of the variables, `filename` or `imagefile`. JRuby can tell that `filename` is a `String` and `imagefile` is a `java.io.File` instance because that's what you assigned them to be.

5 **Read the file into a `BufferedImage` object and create a `Graphics2D` object from it so that you can perform the image processing on it. Add these lines directly after the previous two lines:**

```
bi = javax.imageio.ImageIO.read(imagefile)
w = bi.getWidth()
h = bi.getHeight()
bi2 = BI.new(w, h, BI::TYPE_INT_RGB)
big = bi2.getGraphics()
big.drawImage(bi, 0, 0, nil)
bi = bi2
biFiltered = bi
```

Refer to The Java Tutorial for more information on the Java 2D API.

The important points are :

- You can call Java methods in pretty much the same way in JRuby as you do in Java code.
- You don't have to initialize any variables.
- You can just create a variable and assign anything to it. You don't need to give it a type.

6 **Add the following code to convert the image to gray scale:**

```
colorSpace = java.awt.color.ColorSpace.getInstance(
java.awt.color.ColorSpace::CS_GRAY)
op = java.awt.image.ColorConvertOp.new(colorSpace, nil)
dest = op.filter(biFiltered, nil)
big.drawImage(dest, 0, 0, nil);
```

7 **Stream the file to the browser:**

```
os = java.io.ByteArrayOutputStream.new
javax.imageio.ImageIO.write(biFiltered, "jpeg", os)
string = String.from_java_bytes(os.toByteArray)
send_data string, :type => "image/jpeg", :disposition => "inline",
    :filename => "newkids.jpg"
```

Sometimes you need to convert arrays from Ruby to Java code or from Java code to Ruby. In this case, you need to use the `from_java_bytes` routine to convert the bytes in the output stream to a Ruby string so that you can use it with `send_data` to stream the image to the browser. JRuby provides some other routines for converting types, such as `to_java` to convert from a Ruby Array to a Java String. See Conversion of Types.

▼ To Run a Rails Application That Uses Java 2D Code

1. **Deploy the application on the GlassFish v3 Gem:**

   ```
   asadmin deploy imageprocess
   ```

2. **Run the application by entering the following URL into your browser:**

   ```
   http://localhost:3000/home/index
   ```

   You should now see an image and a link that says, "Perform a ColorConvertOp on this image."

3. **Click the link.**

   You should now see a grayscale version of the image from the previous page.

Monitor Rails Applications on Enterprise Server v3

The Enterprise Server v3 offers monitoring services for various objects.

Monitoring is the process of reviewing the statistics of a system to improve performance or solve problems. The monitoring service can track and display operational statistics, such as the number of requests per second, the average response time, and the throughput. For more information on monitoring, see Chapter 8, "Administering the Monitoring Service," in *Sun GlassFish Enterprise Server v3 Administration Guide*.

Monitoring for JRuby Container

The JRuby Container installed on the Enterprise Server can be configured to be monitored. By default the monitoring services are disabled.

Monitoring can be configured for the container using the following `asadmin` CLI command:

```
configure-jruby-container --monitoring=true
```

The monitoring service can be enabled or disabled. Enable monitoring for GlassFish JRuby container with the following runtime command:

```
enable-monitoring --modules jruby-container=HIGH
```

The other possible values are `OFF` or `LOW`. Disable the monitoring for JRuby container with the following command:

```
disable-monitoring --modules jruby-container=OFF
```

Viewing JRuby Container Statistics

JRuby statistics are available from `asadmin` CLI as well as from Administration Console. To learn more about different types of JRuby statistics available from Enterprise Server, see "JRuby Statistics" in *Sun GlassFish Enterprise Server v3 Administration Guide*.

GlassFish v3 Gem

Another way to work with JRuby is to install the GlassFish v3 Gem on a JRuby standalone instance. The GlassFish v3 Gem is just a lightweight version of the Sun GlassFish Enterprise Server v3 that runs as part of JRuby instance.

When you install the GlassFish v3 Gem on JRuby, you have a Sun GlassFish instance embedded in the JRuby virtual machine. This gives you a more complete development environment because you have everything you need for JRuby on Rails applications running inside the JRuby virtual machine in addition to everything you need from the GlassFish Server to create web applications.

▼ To Install the GlassFish v3 Gem

To install the GlassFish v3 Gem on your JRuby standalone instance, use the following procedure:

1. **Download and install JRuby according to the instructions in "To Install JRuby as Standalone" on page 16.**

2. **Install Rails Gem according to the instructions in "To Install Rails Gem on JRuby" on page 17**

3. **Run the Gem installer to install the GlassFish v3 Gem:**

   ```
   jruby -S gem install glassfish
   ```

4. **Start your server with the following command:**

   ```
   jruby -S glassfish
   ```

▼ To Run a Rails Application on GlassFish v3 Gem

You can also run your JRuby on Rails application on the embedded GlassFish v3 Gem. Create a simple Rails application as described in "Creating a Simple Rails Application" on page 19. Instead of deploying it to the Enterprise Server, use the GlassFish v3 Gem to run that application as described in the following task.

1 **Change to the directory where a sample application has been created. For example:**

```
cd /jruby-apps
```

2 **Deploy the previously created `hello` application:**

```
jruby -S glassfish helloV3
```

3 **Run the application using the following URL in your web browser:**

```
http://localhost:3000/home/index
```

You should now see the following message in your browser window:

```
Welcome to JRuby on Rails on the Sun GlassFish Enterprise Server!
```

Note that the GlassFish v3 Gem runs on port 3000 and not the default port 8080.

▼ To Deploy and Run the Database-Backed Web Application

Creating database-backed web applications was described in "To Create a Database-Backed Rails Application" on page 23. The following task describes running the created `books` application on the GlassFish v3 Gem. You can alternatively deploy it to your regular Enterprise Server using directory-based deployment, as described in "To Deploy the Rails Application as a Directory" on page 22.

1 **Change to `/books` directory.**

2 **Deploy the application to the GlassFish v3 Gem by running the following command:**

```
jruby -S glassfish books
```

3 **Run the application in your web browser using the following URL:**

```
http://localhost:3000/books
```

The opening page says "Listing books" and has an empty table, meaning that there are no book records in the database yet. To add book records to the table, do the next step.

4 **Add records to the table by clicking the New book link on the `index.html` page.**

5 **Enter the data for book on the `new.html` page and click Create.**

Using GlassFish v3 Gem CLI

The GlassFish v3 Gem offers several commands to configure the GlassFish server. You can change the configuration options either by using GlassFish command line or by modifying the `glassfish.yml` file. You view these options by using the following command:

```
glassfish -h
```

The options include `runtimes`, `runtimes-min`, `runtimes-max`, and also a configuration option to create a configuration file for GlassFish Server.

You can create a `glassfish.yml` file which is a default GlassFish configuration file generated by the following command:

```
jruby -S gfrake config
```

Introduction to Warbler

In "Deploying and Running a Rails Application" on page 22, direct deployment of a rails application to Enterprise Server has been described. Warbler provides an easier way to deploy a rails application to a Java application server.

What Is Warbler ?

Warbler is a Gem that makes `WAR` file out of a Rails, Merb, or Rack-based application. Warbler provides a minimal, flexible, Ruby-like way to bundle application files for deployment to a Java application server.

Warbler provides a set of out-of-the box defaults to allow most Rails applications to assemble and work without external Gem dependencies.

Warbler bundles JRuby and the JRuby-Rack Servlet adapter for dispatching requests to the application inside the Java application server, and assembles all jar files in the *WARBLER_HOME*`/lib/` directory into the application.

To learn more about Warbler, see Warbler.

Creating and Deploying a Simple Rails Application with Warbler

The procedure for creating a simple Rails application for Warbler, is similar to the procedure described in "Creating a Simple Rails Application" on page 19.

▼ To Create a Rails Application

1. **Create a new directory for the Warbler application. For example:**

   ```
   mkdir rails-warbler
   ```

2. **Change to `rails-warbler` directory and create a sample application called `hello`:**

   ```
   jruby -S rails hello
   ```

3. **Edit the `enviroment.rb` file to indicate that your application does not use a database:**

   Open `rails-warbler/hello/config/environment.rb` in a text editor.

4. **Remove the pound character (#) in front of line 21 to uncomment it so that it reads as follows and save:**

   ```
   config.frameworks -= [ :active_record, :active_resource, :action_mailer ]
   ```

5. **Exit and save the file.**

6. **Use Warbler to create a war file in `rails-warbler/hello` application directory:**

   ```
   jruby -S warble
   ```

   This creates a `hello.war` file in the directory.

▼ To Deploy the `WAR` File

1. **Change to the `rails-warbler/hello` application directory.**

2. **Deploy the application `WAR` file to the Enterprise Server by running the `asadmin` command:**

   ```
   asadmin deploy hello.war
   ```

3. **Run the `hello` application by entering the following URL in your browser:**

   ```
   http://<hostname>:<port>/hello
   ```

Further Information

For more information on Ruby-on-Rails, JRuby, JRuby on Enterprise Server and Warbler, see the following resources:

- Ruby-on-Rails
- JRuby
- Everything on Scripting in Glassfish
- Warbler

CHAPTER 2

Developing Grails Applications

This chapter introduces Groovy and Grails, a Java technology based alternative to scripting.

The following topics are addressed here:

- "Introduction to Groovy and Grails" on page 35
- "Installing Grails" on page 35
- "Creating a Simple Grails Application" on page 36
- "Deploying and Running a Grails Application" on page 37

Introduction to Groovy and Grails

Groovy is a dynamic, object-oriented language for the Java Virtual Machine, which builds on the strengths of Java but has additional features inspired by languages such as Python, Ruby, and Smalltalk. For more information about Groovy, see Groovy (`http://groovy.codehaus.org`).

Grails is an open source web application framework that leverages the Groovy language and complements Java web development. Grails is a standalone development environment that can hide all configuration details or allow integration of Java business logic. It provides easy-to-use tools to build web applications in Groovy. For more information about Grails, see Grails (`http://www.grails.org`).

Installing Grails

Grails is available as an IPS package from GlassFish Update Center. To develop and deploy Grails applications on the Enterprise Server, you must first install the Grails module.

▼ To Install the Grails Module

1. **Install the Grails add-on component that is available from the Update Tool.**

   For information about the Update Tool, see the *Sun GlassFish Enterprise Server v3 Installation Guide*.

2. **Create a `GRAILS_HOME` environment variable that points to the Grails directory,** *as-install*`/grails`.

3. **Add the** *as-install*`/grails/bin` **directory to the `PATH` environment variable.**

Example 2–1 Setting UNIX Environment Variables

On Solaris, Linux, and other operating systems related to UNIX, use the following commands for Steps 2 and 3 from the above procedure:

```
set GRAILS_HOME=as-install/glassfish/grails
export GRAILS_HOME
cd $GRAILS_HOME
set PATH=$GRAILS_HOME/bin:$PATH
export PATH
chmod a+x $GRAILS_HOME/bin/*
```

Example 2–2 Setting Windows Environment Variables

On the Windows operating system, use the following commands for Steps 2 and 3 from the above procedure:

```
set GRAILS_HOME=C:\GlassFish\grails
set PATH=%GRAILS_HOME%\bin;%PATH%
```

Creating a Simple Grails Application

To create and run a simple `helloworld` application, perform the following tasks:

- "To Create the `helloworld` Application" on page 37
- "To Create the `hello` Controller" on page 37

For more information on creating Grails applications, see the Grails Quick Start (`http://grails.org/Quick+Start`).

▼ To Create the `helloworld` Application

1. **Change to the *as-install*`/grails/samples` directory.**
2. **Run the `grails create-app helloworld` command.**

   The `grails create-app` command creates a `helloworld` application that you can modify.

▼ To Create the `hello` Controller

1. **Change to the *as-install*`/grails/samples/helloworld` directory.**
2. **Run the `grails create-controller hello` command.**

   The `grails create-controller` command creates a controller file that you can modify in the `/grails/samples/helloworld/grails-app/controllers` directory:
3. **Edit the generated `HelloController.groovy` file so that it looks as follows:**

```
class HelloController {

       def world = {
                render "Hello World!"
        }
    //def index = { }
}
```

Deploying and Running a Grails Application

To deploy and run your application, perform the following task:

- "To Run a Grails Application Using Standard Deployment" on page 37

▼ To Run a Grails Application Using Standard Deployment

1. **Change to the application directory. For example:**

   `cd` *as-install*`/grails/samples/helloworld`
2. **Create the WAR file using the following command:**

```
grails war
```

   This command creates a WAR file, `helloworld-0.1.war` in the `helloworld` application directory. The WAR file contains all the application's dependencies, and various jar files.

3 **Deploy the WAR file in one of the following ways:**

- **In the Administration Console, open the Applications component, go to the Web Applications page, select the Deploy button, and type the path to the WAR file. For example:**

  as-install`/grails/samples/helloworld/helloworld-0.1.war`

- **Use the `asadmin deploy` command from command line to deploy the WAR file. For example:**

  ```
  asadmin deploy helloworld-0.1.war
  ```

  Note – Depending on the configuration, you might be prompted for the `asadmin` password at this time.

  ---

4 **To test your application, enter `http://`*host*`:`*port*`/`*war-file-name* in your browser. Do not include the `.war` extension. For example:**

```
http://localhost:8080/helloworld-0.1
```

You should see a screen that shows the following message:

```
Welcome to Grails
```

Clicking the `HelloController` link should change the display to the following message:

```
Hello World!
```

See Also For details about the Administration Console, see the online help.

For details about the `asadmin deploy` command, see the *Sun GlassFish Enterprise Server v3 Reference Manual*.

For details about the grails commands, see the Grails Quick Start (`http://grails.org/Quick+Start`).

For general information about deployment, see the *Sun GlassFish Enterprise Server v3 Application Deployment Guide*.

◆ ◆ ◆ CHAPTER 3

Jython on Django

This chapter provides an overview of Jython and Django and how to get started using them on Sun GlassFish Enterprise Server.

The following topics are addressed here:

Overview

Jython is a Java implementation of the Python language. Jython is integrated with the Java platform and generates the code that runs on Java. Jython implements almost all modules of Python, except those written in C. Jython programs can import and use Java classes effortlessly. Jython provides the following advantages:

- Provides the advantages of easy and powerful Python syntax
- Allows the import of Java classes and extending those classes
- Provides the ability to compile programs to Java bytecode

To learn more about Jython, see the Jython project site.

Django is a web framework for Python and implementations of Python such as Jython. Django allows quick and easy creation of high-performance web applications. Django provides the following advantages:

- Provides an automatic administrative interface to web applications
- Provides an extensible and powerful templating system
- Allows building of data models that can access the databases quickly

To learn more about Django, see the Django project site.

You have the advantages of both Jython and Django when you build web applications using Jython on Django for the Enterprise Server.

Installing Jython and Django

To develop Jython on Django applications for Enterprise Server, you must do the following :

- Install Jython
- Install Django
- Install the Jython container for Enterprise Server

Jython can be installed in one of the following ways:

- Install Jython as a standalone product
- Install Jython from GlassFish Update Center

The following sections explain these tasks in more detail.

▼ To Install Jython as Standalone

You can download Jython and install it as a standalone product. If Jython is installed as standalone, you need to inform the Jython install location to the Enterprise Server.

1. **Download Jython from the following location:**

   http://downloads.sourceforge.net/project/jython/jython/2.5.1/jython_installer-2.5.1.jar

2. **Run the installer as follows:**

   ```
   java -jar jython_installer-2.5.1.jar
   ```

3. **Set the following environmental variables:**

   Set the *JYTHON_HOME* variable to the Jython install location:

   ```
   export JYTHON_HOME=/jython-install-location
   ```

   Add the *JYTHON_HOME*/bin directory to the path:

   ```
   export PATH=$JYTHON_HOME/bin:$PATH
   ```

   You should now be able to invoke Jython from command line as follows:

   ```
   jython
   ```

4. **Inform and configure the Enterprise Server with the location of Jython installation with the following command:**

   ```
   asadmin deploy --property jython.home=/jython-install-location
   ```

▼ To Install Jython from Update Center

The GlassFish Update Center provides a Jython package. The package installs a Jython instance that enables creation of Jython applications for the Enterprise Server. If you installed Jython from Update Center, there is no need for further configuration of `jython.home` property.

1. **Start the Update Tool with the following command:**

   as-install`/bin/updatetool`

2. **Choose the following option from Available Add-ons and click Install:**

   ```
   Jython Runtime IPS package for GlassFish v3
   ```

   Update center automatically completes the installation of the container and configures it for use with Enterprise Server.

3. **Set the following environmental variables:**

   Set the *JYTHON_HOME* variable to the Jython install location:

   `export JYTHON_HOME=/`*as-install*`/glassfish/jython`

   Add the *JYTHON_HOME*/bin directory to the path:

   ```
   export PATH=$JYTHON_HOME/bin:$PATH
   ```

   You should now be able to invoke Jython from command line as follows:

   ```
   jython
   ```

▼ To Install Django

1. **Download Django from the following location:**

   `http://media.djangoproject.com/releases/1.1.1/Django-1.1.1.tar.gz`.

2. **Extract the tar file:**

   ```
   gunzip Django-1.1.1.tar.gz
   ```

   ```
   tar -xvf Django-1.1.1.tar
   ```

3. **Change to the extracted directory:**

   ```
   cd Django-1.1.1
   ```

4. **Install Django with the following command:**

   ```
   jython setup.py install
   ```

▼ To Install Jython Container for Sun GlassFish Enterprise Server

The GlassFish Update Center provides the Jython container package. With this task you can install Jython Container module and Grizzly adapter JAR files in the *as-install*/`glassfish/modules` directory, and enable deployment of Jython/Django applications on the Enterprise Server.

Note – Make sure that the `asadmin` command is available from the PATH variable. Alternately you can use the *as-install*/`bin/asadmin` command.

---

1 **Start the Update Center with the following command:**

as-install/`bin/updatetool`

2 **Choose the following option from Available Add-ons and click Install:**

```
GlassFish V3 Jython Container
```

Update center automatically completes the installation of the container and configures it for use with Enterprise Server.

3 **Start the Enterprise Server:**

```
asadmin start-domain -v
```

4 **Test the configuration with the following process:**

a. **Change to Django examples directory:**

```
cd Django-1.1.1/samples
```

b. **Deploy the example applications on the server:**

```
asadmin deploy .
```

c. **Access the deployed example applications from browser:**

```
http://localhost:8080/examples
```

▼ To Install Jython Support Libraries for Django

The `django-jython` project created database back-ends and management commands for Django and Jython application development. With this task you can install the `django-jython` packages and enable database support for Django.

1 **Download the** `django-jython` **packages from the following location:**

`http://django-jython.googlecode.com/files/django-jython-1.0.0.tar.gz`

2 **Extract the tar file:**

```
gunzip django-jython-1.0.0.tar.gz
```

```
tar -xvf django-jython-1.0.0.tar
```

3 **Change to the extracted directory:**

```
cd django-jython-1.0.0
```

4 **Install the package:**

```
jython setup.py install
```

Creating and Deploying a Simple Jython Application using Django

After completing the software installations, you are ready to create Jython applications using Django. This section explains how to create a simple application. GlassFish users can deploy Django applications with the directory deployment method.

▼ To Create a Simple Django Project

Django comes with a built-in administration utility. You can enable it to make the process of creating projects easier.

1 **Use the following command to enable the Django administration utility:**

```
alias django—admin-jy="jython jython-install-location/bin/django-admin.py"
```

2 **Change to Django install directory:**

```
cd django-install-location
```

3 **Use the following command to create a project:**

```
django-admin-jy startproject <myproject>
```

▼ To Deploy a Django Application From Command Line

To deploy a Django application from the command line using the `asadmin` command, do the following:

1 **Make sure `JYTHON_HOME` and PATH environmental variables are set.**

2 **Change to the directory containing the project. For example:**

```
cd /tools/jython/projects
```

3 **Use the following command to deploy the application:**

```
asadmin deploy myproject/
```

`asadmin` CLI for Jython

The `asadmin deploy` command allows you to set deployment specific properties for the Jython applications. The following table lists these properties.

TABLE 3–1 Jython Properties

Property	Default Value	Possible Value	Description
`jython.home`	None	Path to a directory	Path to a Jython installation. Required.
`jython.mediaRoot`	None	Path to a directory	Optional parameter containing the path to the location, for server to serve the static files.
`jython.frameworkRoot`	None	Path to a directory	Optional parameter containing the path of the framework being used. Currently supports Django.
`jython.applicationType`	None	String representing application type such as Django	Optional parameter to specify the framework, including non-Django applications.

Use the following command syntax to set these properties:

```
asadmin deploy --property <property>=<value>
```

For example, you can set the `jython.frameworkRoot` property to Django directory as follows:

`asadmin deploy --property jython.frameworkRoot=/`*django-install-location*

These values are persistent in the `domain.xml` file.

Using the Django Administration Utility

The previous sections discussed Jython on Django installation and configuration for Sun GlassFish Enterprise Server. To utilize the Django administration utility for creating applications based on a database, you need a database and the JDBC drivers for Jython to connect to that database. The following steps briefly describe the tasks involved:

1. Install a database such as PostgreSQL.
2. Create a database instance.
3. Install `django–jython` packages for the database connectors, if not already installed.
4. Edit the `settings.py` file and configure the database.
5. Edit the `settings.py` file and configure the administration utility.
6. Add the database drivers to the class path to allow Jython to access the database.
7. Sync the database with the following command:

   ```
   jython manage.py syncdb
   ```
8. Edit the `urls.py` file and uncomment the lines pertaining to the administration utility.
9. Make the stylesheets available to the Jython container:

   ```
   asadmin deploy --property
   jython.mediaRoot=/jython-install-location/Lib/site-packages/django/contrib/admin/
   ```

You can obtain more information on how to install and use databases with Django administration from the following tutorial:

http://weblogs.java.net/blog/vivekp/archive/2009/06/run_django_appl_1.html

Further Information

The following links contains more details on the information provided in this chapter:

- http://docs.djangoproject.com/en/dev/howto/jython/#howto-jython
- http://wiki.python.org/jython/DjangoOnJython

◆ ◆ ◆ CHAPTER 4

Scala and Lift

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages. It is also fully interoperable with Java. For details, see `http://www.scala-lang.org/`.

Lift is an expressive and elegant framework for writing web applications using Scala. Lift stresses the importance of security, maintainability, scalability, and performance, while allowing for high levels of developer productivity. For details, see `http://liftweb.net/`.

The following topic is addressed here:

- "Using Scala and Lift" on page 47

Using Scala and Lift

It is common practice to start a Lift web application using Maven. Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting, and documentation from a central piece of information. For details, see `http://maven.apache.org/`.

To create a new Lift project, use Maven interactively in one of the following three ways:

```
mvn archetype:generate -DarchetypeCatalog=http://scala-tools.org/
```

Or:

```
mvn org.apache.maven.plugins:maven-archetype-plugin:1.0-alpha-7:create \
 -DarchetypeGroupId=net.liftweb                             \
 -DarchetypeArtifactId=lift-archetype-blank                 \
 -DarchetypeVersion=0.7.1                                   \
 -DremoteRepositories=http://scala-tools.org/repo-releases  \
 -DgroupId=__my.liftapp__ -DartifactId=__liftapp__
```

Or:

```
mvn org.apache.maven.plugins:maven-archetype-plugin:1.0-alpha-7:create \
 -DarchetypeGroupId=net.liftweb                             \
 -DarchetypeArtifactId=lift-archetype-basic                 \
 -DarchetypeVersion=0.7.1                                   \
 -DremoteRepositories=http://scala-tools.org/repo-releases  \
 -DgroupId=__my.liftapp__ -DartifactId=__liftapp__
```

After coding your application, build the WAR file using the `mvn package` command. Then deploy the WAR file to the Sun GlassFish Enterprise Server as you would any other web application.

◆ ◆ ◆ CHAPTER 5

PHP

PHP is a popular scripting language that is used mainly for generating dynamic web pages. The PHP interpreter takes PHP code as input and produces web pages as output. It can be used as standalone but more often than not, it is deployed on a server.

The following topic is addressed here:

Enabling PHP on Sun GlassFishEnterprise Server

To enable PHP, deploy the Quercus PHP interpreter to the Enterprise Server as a web module.

▼ To Deploy the Quercus PHP Interpreter to the Enterprise Server

1 **Download the Quercus PHP interpreter from** `http://quercus.caucho.com/`**.**

2 **Deploy the downloaded WAR file to the Enterprise Server.**

3 **To verify that your PHP engine is working, enter the following into your browser:**

```
http://localhost:8080/quercus-4.0.1/
```

This is the default PHP script that is available in the Quercus interpreter.

4 **The Quercus application directory is located at** *domain-dir*`/applications/quercus-4.0.1/`**. Move your PHP application to a subdirectory of the Quercus directory. For example:**

as-install/*domain-dir*`/applications/quercus-4.0.1/myapp/`

5 To verify your PHP application is working, access your application from browser. For example, enter the following into your browser:

```
http://localhost:8080/quercus-4.0.1/myapp/
```

See Also For more information, documentation and examples see the Quercus PHP interpreter (`http://quercus.caucho.com/quercus-3.1/index.xtp`).