Getting Started With JRuby on Rails for Sun GlassFish Enterprise Server v3 Prelude

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

Part No: 820–4926–05
August 2008

081029@21288

Contents

JRuby on Rails in GlassFish Tutorial

This tutorial shows you how to get started using JRuby on Rails on the Sun GlassFish™ Enterprise Server v3 Prelude by covering the following topics:

Introduction to JRuby and Rails on the Sun GlassFish

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

- What is Ruby on Rails?
- What is JRuby?
- The Sun GlassFish Enterprise Server v3 Prelude and the GlassFish v3 Gem

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 a database-backed web application, 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 these:

- 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 Prelude, 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 instance.
- The ability of a Rails application to handle multiple requests.

For more details on these and other advantages of using the GlassFish for your JRuby on Rails applications, see Advantages of JRuby-on-Rails with the GlassFish Application Server.

You have two options for deploying a Rails application on the Sun GlassFish:

- Deploy the application as a directory to the Sun GlassFish Enterprise Server v3 Prelude by using the `asadmin` command.
- Deploy the application as a directory to the GlassFish v3 Gem installed on your JRuby virtual machine.

A Gem is a Ruby package that contains a library or an application. In fact, Rails itself is a Gem that you install on top of JRuby.

One way to work with JRuby on Sun GlassFish is to install the GlassFish v3 Gem on top of your JRuby installation. The GlassFish v3 Gem is just a lightweight version of the Sun GlassFish Enterprise Server v3 Prelude and a Grizzly connector for JRuby.

When you install the Gem, you have a Sun GlassFish instance embedded in the JRuby virtual machine. This gives you a more complete development and production 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 Sun GlassFish to create web applications.

Installing JRuby and Required Gems

To develop and deploy Rails applications on the Sun GlassFish, you need to do the following:

1. Download and install JRuby 1.1.4.
2. Install Rails on top of your JRuby installation.
3. Install the GlassFish v3 Gem if you want to deploy your application to a Sun GlassFish instance running inside your JRuby virtual machine.

▼ Downloading and Installing JRuby 1.1.4

1. **Go to the JRuby download site (`http://dist.codehaus.org/jruby`).**
2. **Download `jruby-bin-1.1.4.zip`.**
3. **Unpack the zip file.**
4. **Set your `JRUBY_HOME` environment variable to the location of your JRuby 1.1.4 installation.**
5. **Add `<JRUBY_HOME>/bin` to your system path so that you can invoke JRuby from anywhere in your directory tree.**

▼ Installing Rails on JRuby

● **Install the Rails Gem:**

```
jruby –S gem install rails
```

You should see the following output, which tells you that six Gems and their documentation have been installed:

```
JRuby limited openssl loaded. gem install jruby-openssl for full support.
http://wiki.jruby.org/wiki/JRuby_Builtin_OpenSSL
Successfully installed activesupport-2.1.1
Successfully installed activerecord-2.1.1
Successfully installed actionpack-2.1.1
Successfully installed actionmailer-2.1.1
Successfully installed activeresource-2.1.1
Successfully installed rails-2.1.1
6 gems installed
Installing ri documentation for activesupport-2.1.1...
Installing ri documentation for activerecord-2.1.1...
Installing ri documentation for actionpack-2.1.1...
Installing ri documentation for actionmailer-2.1.1...
Installing ri documentation for activeresource-2.1.1...
Installing RDoc documentation for activesupport-2.1.1...
Installing RDoc documentation for activerecord-2.1.1...
Installing RDoc documentation for actionpack-2.1.1...
Installing RDoc documentation for actionmailer-2.1.1...
Installing RDoc documentation for activeresource-2.1.1...
```

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.

▼ Installing the GlassFish v3 Gem

One of the ways to deploy a Rails application is to deploy it to the Sun GlassFish instance running inside the JRuby virtual machine. To do this, you have to install the GlassFish v3 Gem on top of your JRuby installation:

● **Run the Gem installer to install the GlassFish v3 Gem:**

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

You should see the following output:

```
JRuby limited openssl loaded. gem install jruby-openssl for full support.
http://wiki.jruby.org/wiki/JRuby_Builtin_OpenSSL
Successfully installed glassfish-0.3.1-universal-java
1 gem installedSuccessfully installed glassfish-0.1.2-universal-java
1 gem installed
```

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 “Welcome to JRuby on Rails on the Sun GlassFish Enterprise Server!”

▼ Creating the hello Application

1 **Go to** `<JRUBY_HOME>/samples` **directory.**

2 **Create a Rails application called** `hello`**:**

```
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.

▼ Creating the Controller and View

By doing this task, you can 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 **Go to the** `<JRUBY_HOME>/samples/hello` **directory you created in the previous task.**

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

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

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

▼ Passing Data From the Controller to the View

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

1 **Open** `<JRUBY_HOME>/samples/hello/app/controllers/home_controller.rb` **in a text editor.**

2 **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, it makes the `@hello_message` variable available to `index.html.erb`.

3 **Save the file.**

4 **Open** `<JRUBY_HOME>/samples/hello/app/views/home/index.html.erb` **file in a text editor.**

5 **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.

6 **Save the file.**

▼ Using 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 need to edit the `enviroment.rb` configuration file to indicate that your application does not use a database.

1 **Open** `<JRUBY_HOME>/samples/hello/config/environment.rb` **file in a text editor.**

2 **Remove the pound character (#) in front of line 21 to uncomment it so that it reads as:**

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

`ActiveRecord` supports database access for Rails applications. When you create model objects, you will most likely base them on `ActiveRecord::Base`.

3 **Save the file.**

Deploying and Running a Rails Application

As described in "JRuby on Rails, the Sun GlassFish Enterprise Server v3 Prelude, and the GlassFish v3 Gem" on page 6, you have two ways to deploy your Rails application on the Sun GlassFish:

- Deploy it natively as a directory using the `asadmin` command.
- Deploy it using the GlassFish v3 Gem.

This section shows you how to deploy the hello application you created in the previous section natively and with the GlassFish v3 Gem and how to run the application in your web browser. You can also use these same instructions to deploy a legacy Rails application.

▼ Deploying a Rails Application as a Directory

You can use directory-based deployment to deploy any Rails application natively to the Sun GlassFish Enterprise Server. To natively deploy the `hello` application to the Enterprise Server:

1 **Set `JRUBY_HOME` property value to the path to your JRuby installation as the last line in one of the following files, located in the `config` directory of Enterprise Server your installation:**

- **For Windows systems:** `asenv.bat`
- **For Unix systems:** `asenv.conf`

2 **Save the file.**

3 **Edit `<AS_INSTALL>/domains/domain1/config/domain.xml`and add this entry inside element:**

```
<java-config>
...
<jvm-options>-Djruby.home=<JRUBY_HOME></jvm-options>
...
</java-config>
```

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

---

4 **Start the server.**

5 **Go to `<JRUBY_HOME>/samples`.**

6 **Deploy the `hello` application with `asadmin` command from your Enterprise Server installation:**

```
<AS_INSTALL>/bin/asadmin deploy hello
```

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

```
http://localhost:8080/hello/home/index
```

▼ Deploying a Rails Application to the GlassFish v3 Gem

1 **Go to `<JRUBY_HOME>/samples`.**

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

```
jruby -S glassfish_rails helloV3
```

When the GlassFish instance is finished launching, you should see output similar to the following:

```
INFO: Rails instance instantiation took : 37754ms
```

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!
```

Notice that the GlassFish v3 Gem runs on port 3000, not 8080.

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™. It is assumed that you have already installed JRuby 1.1.4, Rails 2.1.1, and the required Gems.

▼ Setting Up the MySQL Database Server

1 **Download and install the MySQL 5.0 Community Server (`http://dev.mysql.com/downloads/mysql/5.0.html#downloads`)**

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

3 **Start the server.**

▼ Creating a Database-Backed Rails Application

1. **Go to the `<JRUBY_HOME>/samples` directory of your JRuby 1.1.4 installation.**
2. **Create the `books` application template so that it is configured to use the MySQL database:**

   ```
   jruby -S rails books -d mysql
   ```
3. **Go to the `books` directory you just created.**
4. **Open the `config/database.yml` file in a text editor.**
5. **When prompted, enter your MySQL root password under the development heading in the `database.yml` file.**
6. **Go back to the `books` directory if you are not already there.**
7. **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.
8. **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.
9. **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 `jruby –S rake db:reset`.

▼ Deploying and Running the Database-Backed Web Application

With this task, you will deploy the `books` application to the GlassFish v3 Gem. You can alternatively deploy it to your regular Enterprise Server using directory-based deployment, as described in "Deploying a Rails Application as a Directory" on page 11.

1. **Go to** `<JRUBY_HOME>/samples/books`**.**

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

   ```
   jruby -S glassfish_rails 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.**

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.

Creating the Rails Application That Accesses Java Libraries

1. **Go to** `<JRUBY_HOME>/samples`**.**
2. **Create an application by running this command:**

   ```
   jruby -S rails imageprocess
   ```
3. **Open the** `<JRUBY_HOME>/samples/imageprocess/config/environment.rb` **file in a text editor.**
4. **Follow steps 2 and 3 from the instructions in section, "Using Rails Without a Database" on page 10.**
5. **Go to the** `<JRUBY_HOME>/samples/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. **Go to the** `<JRUBY_HOME>/samples/imageprocess/app/views/home` **directory.**
8. **Create a second view by copying the default view into a view called** `seeimage.html.erb`**:**

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

Creating 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 on it.
- 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** `<JRUBY_HOME>/samples/imageprocess/public/image` **file.**

3 **Go to** `<JRUBY_HOME>/samples/imageprocess/app/views/home` **file.**

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 `<JRUBY_HOME>/samples/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** `index.html.erb` **file.**

8 **Open** `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 you loaded into `index.html.erb`.

▼ Adding 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 grayscale:**

```
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.

▼ Running a Rails Application That Uses Java 2D Code

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

```
jruby -S glassfish_rails 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.

Configuring JRuby Runtime Pool

The Sun GlassFish Enterprise Server v3 Prelude 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.

JRuby runtime pool is configured in `<AS_INSTALL>/domains/domain1/config/domain.xml` file. Edit the file and add the elements below with the values that you want the runtime to be configured with as indicated.

```
<java-config>
<jvm-options>-Djruby.runtime.min=1</jvm-options>
<jvm-options>-Djruby.runtime=2</jvm-options>
<jvm-options>-Djruby.runtime.max=3</jvm-options>
</java-config>
```

The properties of the above elements are explained as follows:

- *-Djruby.runtime=X* sets the initial number of JRuby runtimes that GlassFish starts with. The default value is one. This represents the highest value that GlassFish accepts as minimum runtimes, and the lowest value that GlassFish uses as maximum runtimes.
- *-Djruby.runtime.max=X* sets the maximum number of JRuby runtimes that might be available in the pool. The default value is two. For this element, too high values might result in `OutOfMemory` errors, either in the heap or in the `PermGen`.
- *-Djruby.runtime.min=X* 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.

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.

Introduction to Warbler

In "Deploying and Running a Rails Application" on page 11, we have seen how to deploy a rails application directly to Enterprise Server. 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 `<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 9.

▼ Creating a Rails application

1 **Create a new directory under `<JRUBY_HOME>/samples` directory called `rails-warbler`.**

2 **Go to `<JRUBY_HOME>/samples/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 `<JRUBY_HOME>/samples/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 **Use Warbler to create a war file in `<JRUBY_HOME>/samples/rails-warbler/hello` application directory:**

```
jruby -S warble
```

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

▼ Deploying the war file

1 **Go to the application directory `<JRUBY_HOME>/samples/rails-warbler/hello`.**

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

```
<AS_INSTALL>/bin/asadmin deploy hello.war
```

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

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

Further Information

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

- Ruby-on-Rails
- JRuby
- Scripting on the GlassFish Application Server
- Warbler