**Progoti Systems Limited**

12

Technical Instruction

GETTING STARTED WITH 'Grails 2.x'

**Md. Afzalur Rashid**

Contents

[GETTING STARTED WITH 'Grails 2.x' 1](file:///D:\Afzal\Progoti%20Officials\SVN-Repository\Agrani%20School\FinanceManagementSystem\docs\TechnicalInstruction-Grails_V-1.1.docx#_Toc343682375)

[1. Introductions 4](#_Toc343682376)

[2. Install and Configure JDK 4](#_Toc343682377)

[2.1 Download 4](#_Toc343682378)

[2.2 Installation 4](#_Toc343682379)

[2.2.1 Step 1: Download and run jdk-6u31-windows-x64.exe 4](#_Toc343682380)

[2.2.2 Step 2: Choose the installation folder & packages 6](#_Toc343682381)

[2.2.3 Step 3: Wait for the jdk installation processing 7](#_Toc343682382)

[2.2.4 Step 4: Choose the destination folder for jre installation 8](#_Toc343682383)

[2.2.5 Step 5: Wait for the installation to complete 9](#_Toc343682384)

[2.3 Configure Environment Variables 10](#_Toc343682385)

[2.3.1 Configure JAVA\_HOME environment variable 14](#_Toc343682386)

[2.3.2 Configure/Edit path environment variable 18](#_Toc343682387)

[3. Grails Basics 20](#_Toc343682388)

[4. Install and Configure GRAILS 20](#_Toc343682389)

[4.1 Download Grails 21](#_Toc343682390)

[4.2 Install Grails 21](#_Toc343682391)

[4.3 Configure Environment Variable for grails 24](#_Toc343682392)

[4.3.1 Configure GRAILS\_HOME environment variable 28](#_Toc343682393)

[4.3.2 Configure/Edit path environment variable 32](#_Toc343682394)

[4.3.3 Tip 34](#_Toc343682395)

[5. Creating grails Application 34](#_Toc343682396)

[5.1 A Hello World Example 34](#_Toc343682397)

[6. Getting Set Up in an IDE 36](#_Toc343682398)

[6.1 IntelliJ IDEA 36](#_Toc343682399)

[6.2 Eclipse 36](#_Toc343682400)

[7. Convention over Configuration 37](#_Toc343682401)

[8. Running an Application 37](#_Toc343682402)

[9. Testing an Application 38](#_Toc343682403)

[10. Deploying an Application 38](#_Toc343682404)

[11. More detail domain based project creation 39](#_Toc343682405)

[11.1 Create application 39](#_Toc343682406)

[11.1.1 Create new project 39](#_Toc343682407)

[11.1.2 Create domain class 39](#_Toc343682408)

[11.2 Create project in IntelliJ Idea 39](#_Toc343682409)

[11.2.1 Open IntelliJ Idea 39](#_Toc343682410)

[12 Web Services 46](#_Toc343682411)

[12.1 REST 46](#_Toc343682412)

[12.2 REST with JAX-RS (Jersey) 50](#_Toc343682413)

# 1. Introductions

**Grails** is an open source web application framework which uses the [Groovy](http://en.wikipedia.org/wiki/Groovy_%28programming_language%29) programming language (which is in turn based on the [Java platform](http://en.wikipedia.org/wiki/Java_%28Sun%29#Platform)). It is intended to be a high-productivity framework by following the ["coding by convention"](http://en.wikipedia.org/wiki/Convention_over_configuration) paradigm, providing a stand-alone development environment and hiding much of the configuration detail from the developer.

Grails is a dynamic web application framework built on Java and Groovy, leveraging best of breed APIs including Spring, Hibernate and SiteMesh. Grails brings to Java and Groovy developers the joys of convention-based rapid development while allowing them to leverage their existing knowledge and capitalize on the proven and performant APIs Java developers have been using for years.

Grails was previously known as 'Groovy on Rails'; in March 2006 that name was dropped in response to a request by [David Heinemeier Hansson](http://en.wikipedia.org/wiki/David_Heinemeier_Hansson), founder of the [Ruby on Rails](http://en.wikipedia.org/wiki/Ruby_on_Rails) framework. Work began in July 2005, with the 0.1 release on March 29, 2006 and the 1.0 release announced on February 18, 2008.The latest version is 2.0.3 which is released on April 2012.

Since grails built on Java, I will explain steps for installing both java and grails in windows 7 so that developers can start working on grails without any further difficulty.

# 2. Install and Configure JDK

## 2.1 Download

Download latest version of Java (JDK) form [Oracle website](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

http://www.oracle.com/technetwork/java/javase/downloads/index.html

## 2.2 Installation

There are two ways to install JDK

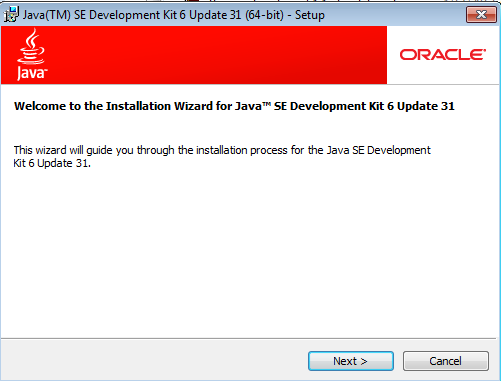
1. Install Binary distribution for windows at your favorite folder (I have installed jdk1.6.0\_23 to C:\Program Files (x86)\Java\jdk1.6.0\_23 folder)\
2. Download plug and play zipped distribution and extract to any folder (say C:\Program Files (x86)\Java\jdk1.6.0\_23 )

The following screenshots will help you to install java sdk:

### 2.2.1 Step 1: Download and run jdk-6u31-windows-x64.exe

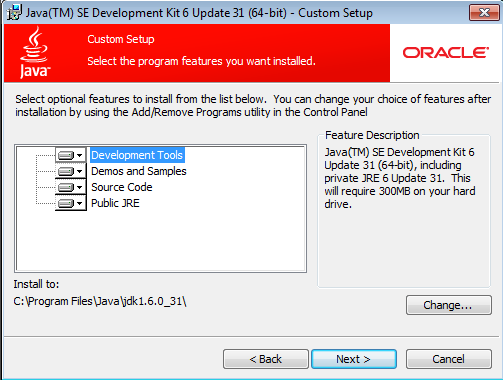
A dialogue will popup and will ask to allow the application installation

Click on Yes button to start the installation



Click on next button

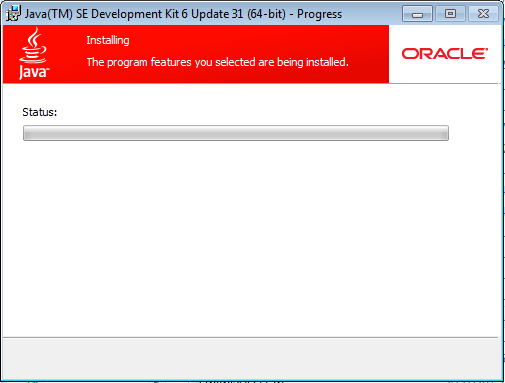
### 2.2.2 Step 2: Choose the installation folder & packages



Choose the installation folder and packages. By default C:\Program Files\Java\jdk1.6.0\_31 will be picked up , you may change this by clicking on Change... button.

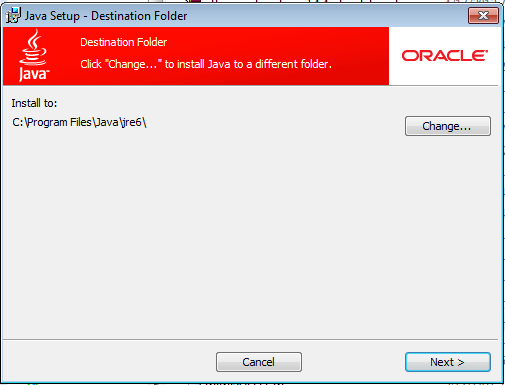
Click on next

### 2.2.3 Step 3: Wait for the jdk installation processing



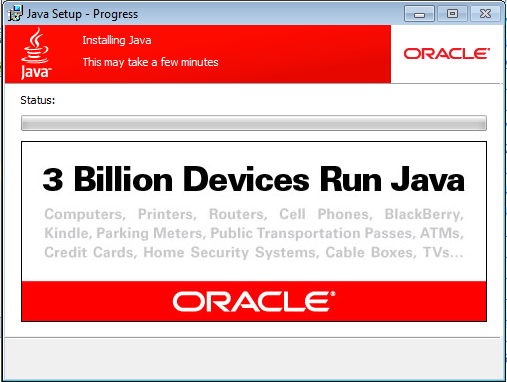
2.2.4 Step 4: Choose the destination folder for jre installation.

By default By default C:\Program Files\Java\jre will be picked up. You may change this by clicking Change... button



Click on Next Button

### 2.2.5 Step 5: Wait for the installation to complete



After the completion of the installation the following screen will be displayed.

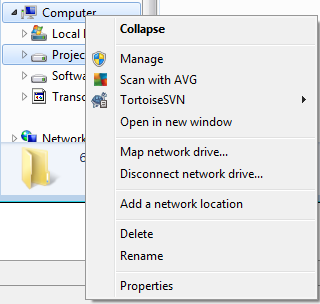


You may Register the product by clicking the 'Product Registration Information' buton. Registration process is free of cost. The installation does not have any impact on registration , so you can just click finish to end the installation process.

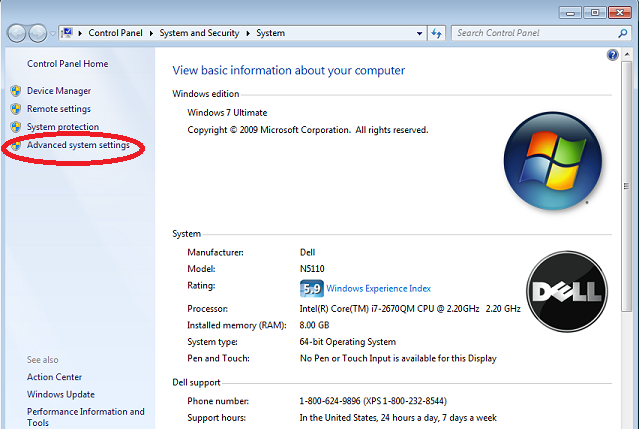
## 2.3 Configure Environment Variables

Now you have to configure environment variable JAVA\_HOME and path to make sure any application looks JDK through these variables.

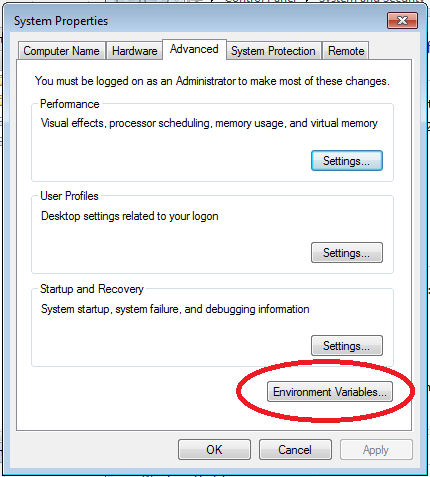
To configure environment variables right click on Computer and then chose properties



Once you click on properties on the previous screen the following screen will appear.



Now click on Advanced System settings. The following screen will appear.



Click on' Environment variable' button to get the window to configure environment variables.

The following System Properties window will appear

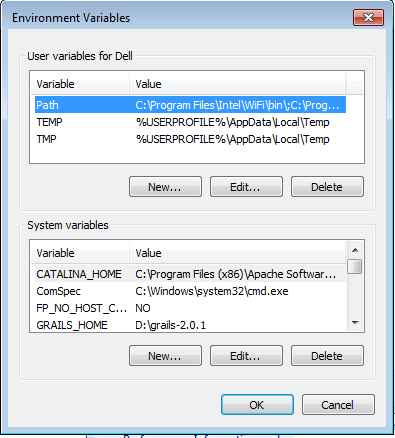
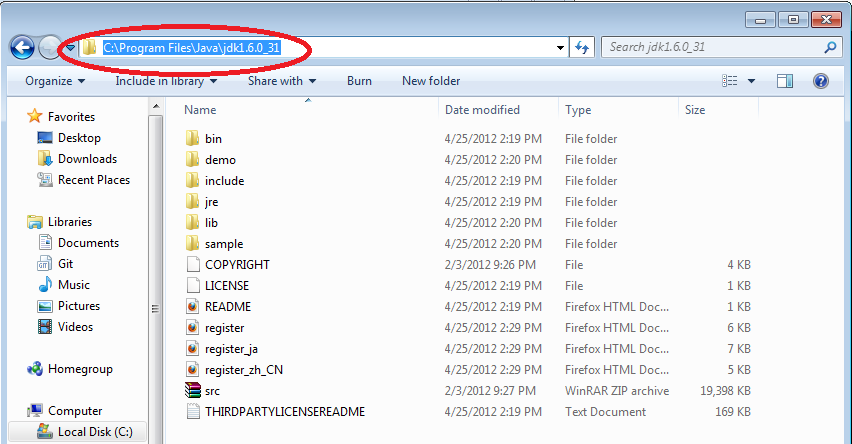


Figure - Environment variables

Remember to setup system variables not user variables. If you setup user variables other users will not get this variables.

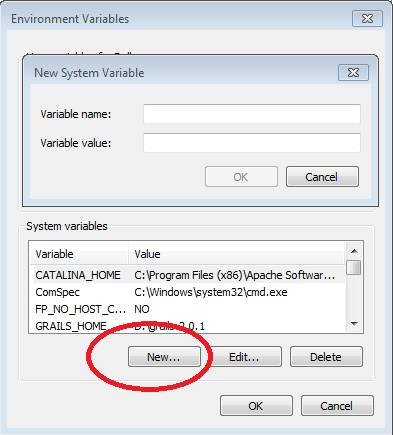
### 2.3.1 Configure JAVA\_HOME environment variable

To configure JAVA\_HOME environment variable copy the location of jdk installation (C:\Program Files\Java\jdk1.6.0\_31). Make sure bin,demo,include,jre etc. folders are available inside this folder



Now click on new in Figure - Environment variables screen

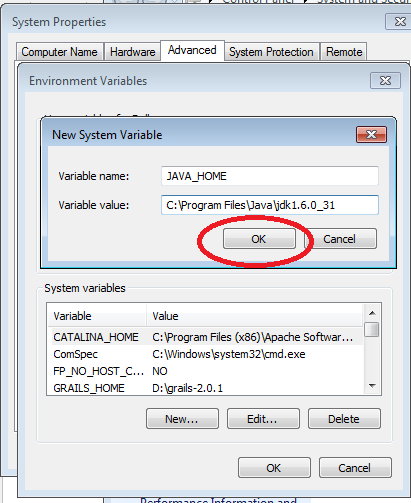
A small popup will appear like the following



Type JAVA\_HOME in variable name text field

and paste the copied location of the jdk installation

help from the following screen

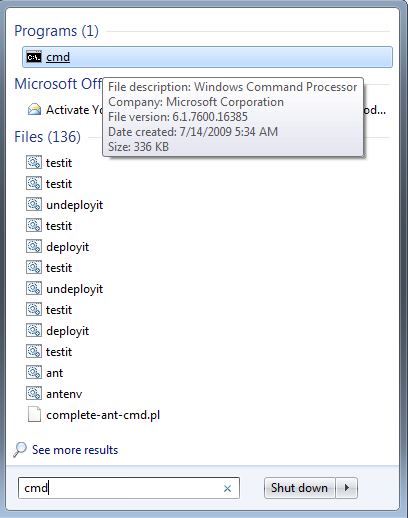


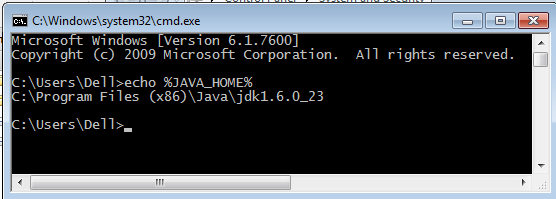
Press ok button.

The environment variable JAVA\_HOME is now configured properly.

To test open a command prompt and type echo %JAVA\_HOME%

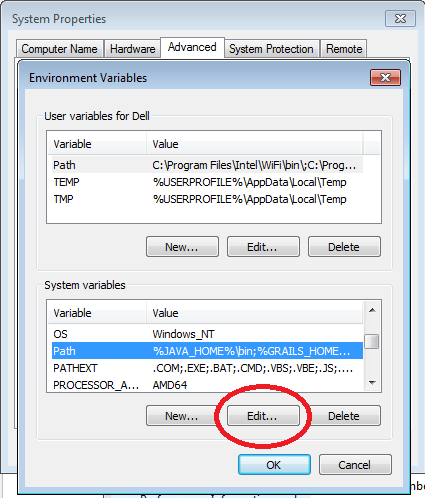
See the following screenshoot





### 2.3.2 Configure/Edit path environment variable

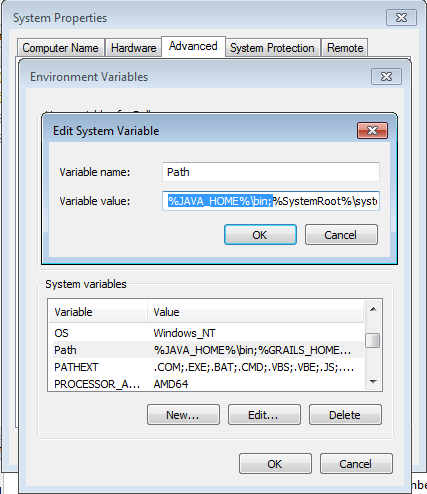
To configure path environment variable select the path environment variable like the following screen. Here the Blue colored like is the selected path environment variable on system variables.



Click on edit button

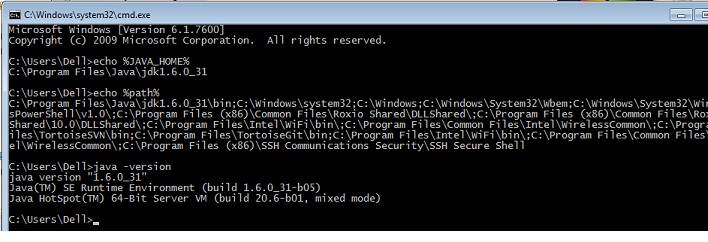
And add %JAVA\_HOME%\bin; at the very beginning of the variable value field.

Follow the screen below:



Now test configuration typing java -version on the command prompt.

follow the below screen shot for more help:

If you can see the java version properly as installed version of java , you are done with java(jdk) installation.

Now JDK is installed properly. You may now download and install Grails for configuration.

# 3. Grails Basics

Grails is a web framework based on [Groovy](http://www.vogella.com/articles/Groovy/article.html) and Java which can be deployed into existing Java web servers, e.g. [Tomcat](http://www.vogella.com/articles/ApacheTomcat/article.html) or [Jetty](http://www.vogella.com/articles/Jetty/article.html).

Grails allows to create quickly web applications; its scaffolding capabilities let you create a new project within a few minutes. Grails is based on the "convention over configuration" idea which allows the application to auto-wires itself based on naming schemes (instead of using configuration files, e.g. XML files).

The Grails framework allows instance development without requiring any configuration. Just download Grails and you are ready to start. Grails accomplish this by automatically providing the Tomcat webcontainer and the H2 database during development. If you deploy you Grails application later you can use another webcontainer or database.

Grails uses GORM (Grails Object Relational Mapping) for the persistence of the domain model. GORM is based on Hibernate. You can test with the H2 and run in production against another database simply by changing the configuration file (DataSource.groovy).

Grails uses JavaEE as the architectural basis and Spring for structuring the application via dependency injection.

Grails is plugin based and provides its own build system (Gant). The Grails homepage provides several pre-defined plugins which extend the Grails framework.

During the start of a new development with Grails you mainly use the command line to generated new user interfaces.

# 4. Install and Configure GRAILS

The first step to getting up and running with Grails is to install the distribution. To do so follow these basic steps:

[Download](http://grails.org/Download) a binary distribution of Grails and extract the resulting zip file to a location of your choice

Setup your GRAILS\_HOME environment variable pointing to your installation directory of Grails. Add also the $GRAILS\_HOME/bin to the path variable.

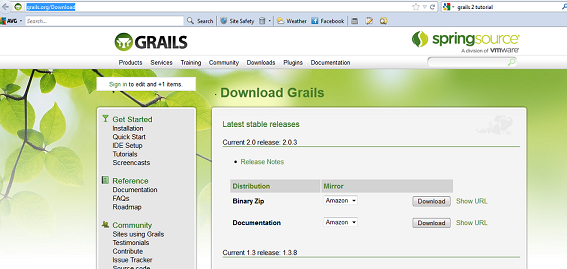
If Grails is working correctly you should now be able to type grails -version in the terminal window and see output similar to this:

Grails version: 2.0.3

Please follow the following steps for more detail on installation and configuration of grails on windows 7 machine.

## 4.1 Download Grails

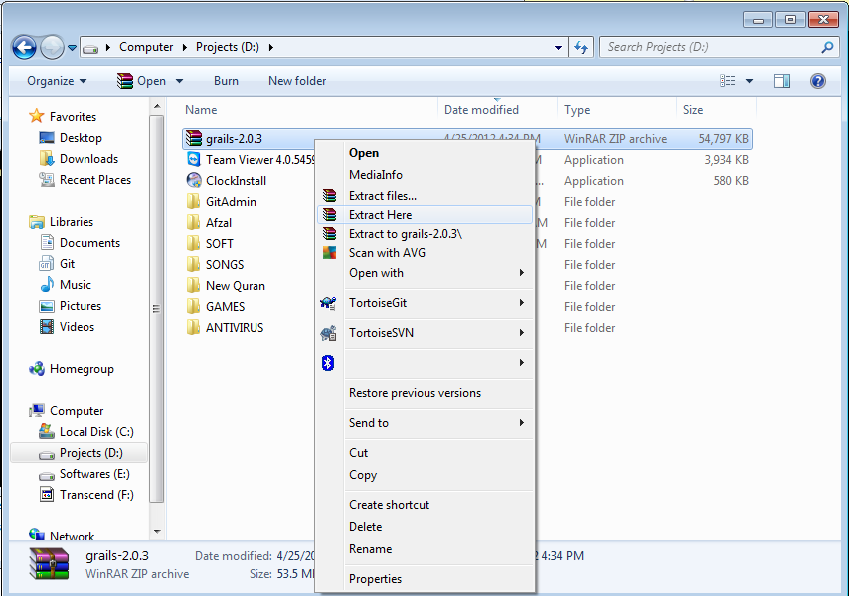
To download grails please visit <http://grails.org/Download>



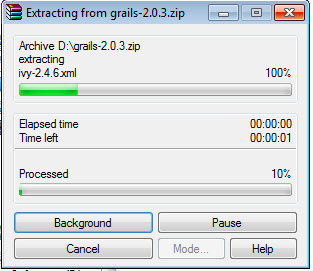
Click the download button of Binary zip. The Download will start. It will take around 20 minutes to download the 57.5 MB download with average internet speed.

## 4.2 Install Grails

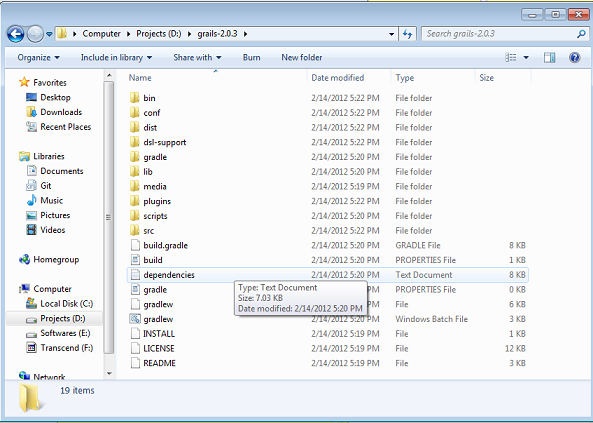
Copy the zip file to D:\\ drive or any directory of your choice. Right click on the file and choose extract here to extract the distribution to the folder where grails-2.0.3.zip folder resides.



In this example the grails-2.0.3.zip is extracted to d:\



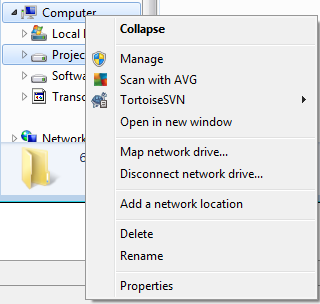
Verify that the folder has bin, conf, dist etc folders. The following screen may help you to verify the distribution.



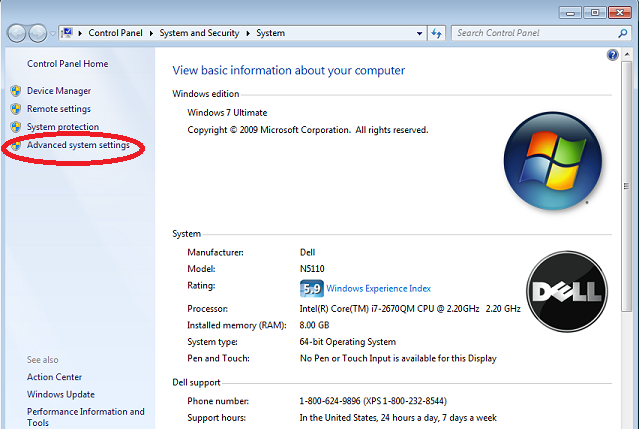
## 4.3 Configure Environment Variable for grails

Now you have to configure environment variable JAVA\_HOME and path to make sure any application looks JDK through these variables.

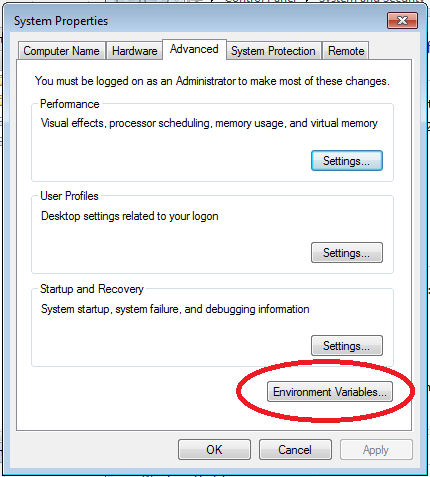
To configure environment variables right click on Computer from windows explorer and then chose properties



Once you click on properties on the previous screen the following screen will appear.



Now click on Advanced System settings. The following screen will appear.



Click on' Environment variable' button to get the window to configure environment variables.

The following System Properties window will appear

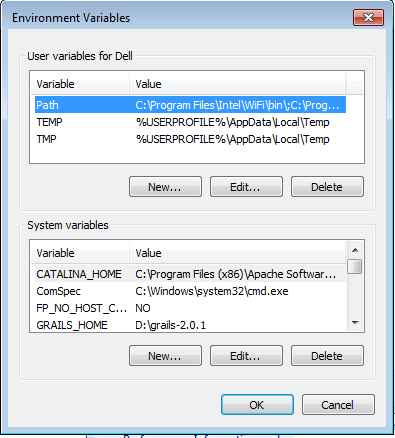
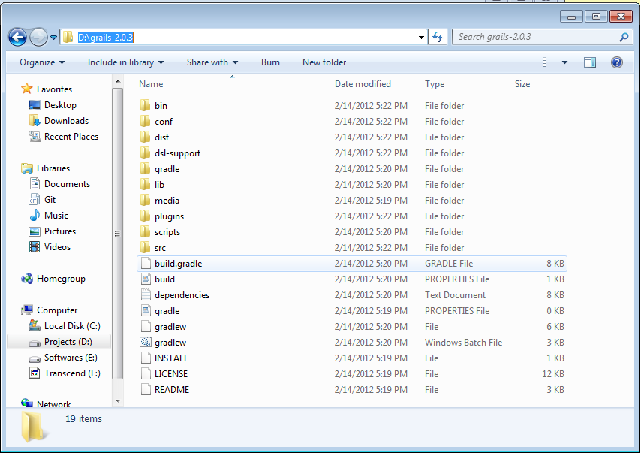


Figure - Grails Environment variables

Remember to setup system variables not user variables. If you setup user variables other users will not get this variables.

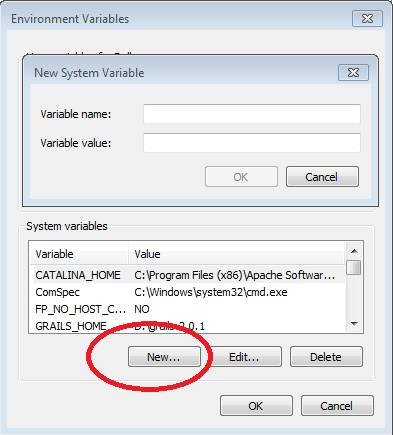
### 4.3.1 Configure GRAILS\_HOME environment variable

To configure GRAILS\_HOME environment variable copy the location of grails installation (D:\grails-2.0.3). Make sure bin, conf, dist etc folders are available inside this folder



Now click on new in Figure - Environment variables screen

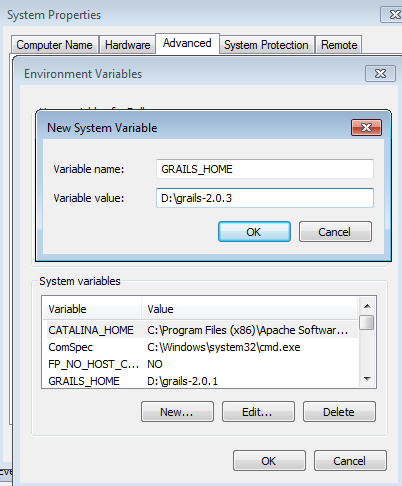
A small popup will appear like the following



Type GRAILS\_HOME in variable name text field

and paste the copied location of the grails installation in variable value field

help from the following screen

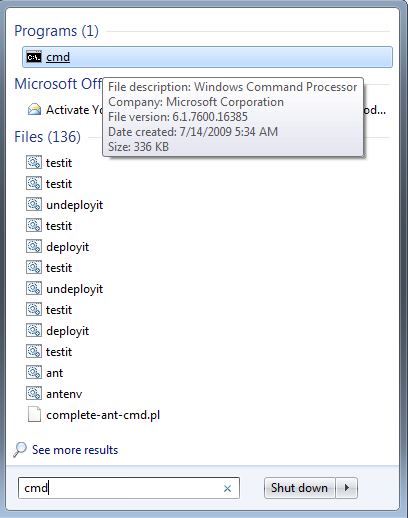


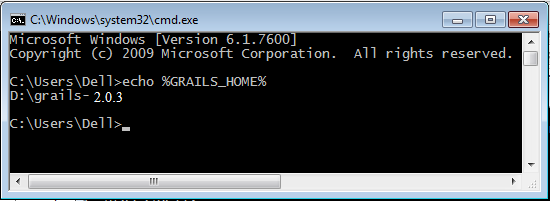
Press ok button.

The environment variable GRAILS\_HOME is now configured properly.

To test open a command prompt and type echo %GRAILS\_HOME%

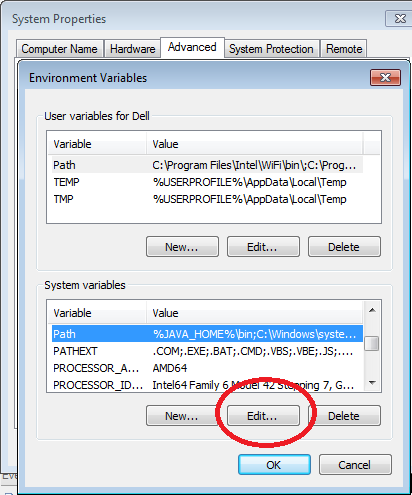
See the following screenshoot





### 4.3.2 Configure/Edit path environment variable

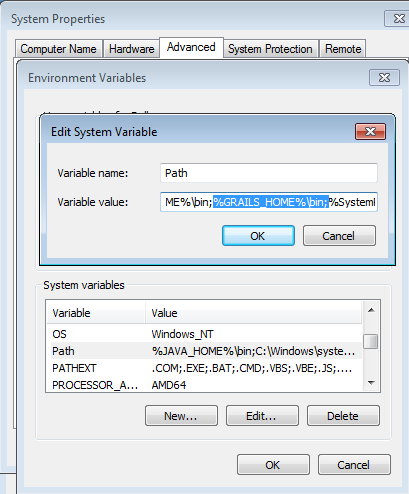
To configure path environment variable select the path environment variable like the following screen. Here the Blue colored like is the selected path environment variable on system variables.



Click on edit button

And add %GRAILS\_HOME%\bin; after %JAVA\_HOME%\bin; of the variable value field.

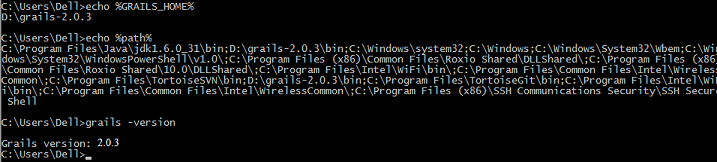
Follow the screen below:



Click on ok button to finish setting path variable

Now test configuration typing java -version on the command prompt.

follow the below screen shot for more help:

If you can see the grails version properly as installed version of grails , you are done with grails installation.

### 4.3.3 Tip

Please make sure that that the environment variable JAVA\_HOME is set to the JDK and not the JRE. The JDK is required to develop with Grails.

# 5. Creating grails Application

To create a Grails application you first need to familiarize yourself with the usage of the grails command which is used in the following manner:

grails [command name]

Run [create-app](http://grails.org/doc/latest/ref/Command%20Line/create-app.html) to create an application:

grails create-app helloworld

This will create a new directory inside the current one that contains the project. Navigate to this directory in your console:

cd helloworld

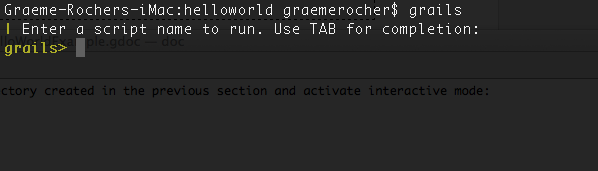
## 5.1 A Hello World Example

Let's now take the new project and turn it into the classic "Hello world!" example. First, change into the "helloworld" directory you just created and start the Grails interactive console:

cd helloworld

grails

You should see a prompt that looks like this:



What we want is a simple page that just prints the message "Hello World!" to the browser. In Grails, whenever you want a new page you just create a new controller action for it. Since we don't yet have a controller, let's create one now with the [create-controller](http://grails.org/doc/latest/ref/Command%20Line/create-controller.html) command:

grails> create-controller hello

Don't forget that in the interactive console, we have auto-completion on command names. So you can type "cre" and then press <tab> to get a list of all create-\* commands. Type a few more letters of the command name and then <tab> again to finish.

The above command will create a new [controller](http://grails.org/doc/latest/guide/single.html#controllers) in the grails-app/controllers/helloworld directory called HelloController.groovy. Why the extra helloworld directory? Because in Java land, it's strongly recommended that all classes are placed into packages, so Grails defaults to the application name if you don't provide one. The reference page for [create-controller](http://grails.org/doc/latest/ref/Command%20Line/create-controller.html) of grails web site provides more detail on this.

We now have a controller so let's add an action to generate the "Hello World!" page. The code looks like this:

package helloworld

class HelloController {

def index() {

render "Hello World!"

}

}

The action is simply a method. In this particular case, it calls a special method provided by Grails to render the page.

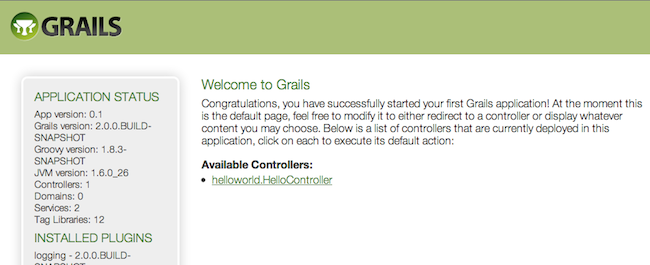
Job done. To see your application in action, you just need to start up a server with another command called run-app:

grails> run-app

This will start an embedded server on port 8080 that hosts your application. You should now be able to access your application at the URL <http://localhost:8080/helloworld/> - try it!

If you see the error "Server failed to start for port 8080: Address already in use", then it means another server is running on that port. You can easily work around this by running your server on a different port using -Dserver.port=9090 run-app. '9090' is just an example: you can pretty much choose anything within the range 1024 to 49151.

The result will look something like this:



This is the Grails intro page which is rendered by the grails-app/view/index.gsp file. It detects the presence of your controllers and provides links to them. You can click on the "HelloController" link to see our custom page containing the text "Hello World!". Great! You have your first working Grails application.

One final thing: a controller can contain many actions, each of which corresponds to a different page (ignoring AJAX at this point). Each page is accessible via a unique URL that is composed from the controller name and the action name: /<appname>/<controller>/<action>. This means you can access the Hello World page via [/helloworld/hello/index](http://localhost:8080/helloworld/hello/index), where 'hello' is the controller name (remove the 'Controller' suffix from the class name and lower-case the first letter) and 'index' is the action name. But you can also access the page via the same URL without the action name: this is because 'index' is the default action .

# 6. Getting Set Up in an IDE

## 6.1 IntelliJ IDEA

[IntelliJ IDEA](http://www.jetbrains.com/idea) and the [JetGroovy](http://www.jetbrains.net/confluence/display/GRVY/Groovy+Home) plugin offer good support for Groovy and Grails developers. Refer to the section on [Groovy and Grails](http://www.jetbrains.com/idea/features/groovy_grails.html) support on the JetBrains website for a feature overview.

To integrate Grails with IntelliJ run the following command to generate appropriate project files:

grails integrate-with --intellij

## 6.2 Eclipse

We recommend that users of [Eclipse](http://www.eclipse.org/) looking to develop Grails application take a look at [SpringSource Tool Suite](http://www.springsource.com/products/sts), which offers built in support for Grails including automatic classpath management, a GSP editor and quick access to Grails commands. See the [STS Integration](http://www.grails.org/STS+Integration) page for an overview.

#### 6.3 NetBeans

NetBeans provides a Groovy/Grails plugin that automatically recognizes Grails projects and provides the ability to run Grails applications in the IDE, code completion and integration with the Glassfish server. For an overview of features see the [NetBeans Integration](http://www.grails.org/NetBeans+Integration) guide on the Grails website which was written by the NetBeans team.

# 7. Convention over Configuration

Grails uses "convention over configuration" to configure itself. This typically means that the name and location of files is used instead of explicit configuration, hence you need to familiarize yourself with the directory structure provided by Grails.

Here is a breakdown and links to the relevant sections:

* grails-app - top level directory for Groovy sources
  + conf - [Configuration sources](http://grails.org/doc/latest/guide/single.html#conf).
  + controllers - [Web controllers](http://grails.org/doc/latest/guide/single.html#controllers) - The C in MVC.
  + domain - The [application domain](http://grails.org/doc/latest/guide/single.html#GORM).
  + i18n - Support for [internationalization (i18n)](http://grails.org/doc/latest/guide/single.html#i18n).
  + services - The [service layer](http://grails.org/doc/latest/guide/single.html#services).
  + taglib - [Tag libraries](http://grails.org/doc/latest/guide/single.html#taglibs).
  + utils - Grails specific utilities.
  + views - [Groovy Server Pages](http://grails.org/doc/latest/guide/single.html#gsp) - The V in MVC.
* scripts - [Gant scripts](http://grails.org/doc/latest/guide/single.html#commandLine).
* src - Supporting sources
  + groovy - Other Groovy sources
  + java - Other Java sources
* test - [Unit and integration tests](http://grails.org/doc/latest/guide/single.html#testing).

# 8. Running an Application

Grails applications can be run with the built in Tomcat server using the [run-app](http://grails.org/doc/latest/ref/Command%20Line/run-app.html) command which will load a server on port 8080 by default:

grails run-app

You can specify a different port by using the server.port argument:

grails -Dserver.port=8090 run-app

Note that it is better to start up the application in interactive mode since a container restart is much quicker:

$ grails

grails> run-app

| Server running. Browse to http://localhost:8080/helloworld

| Application loaded in interactive mode. Type 'exit' to shutdown.

| Downloading: plugins-list.xml

grails> exit

| Stopping Grails server

grails> run-app

| Server running. Browse to http://localhost:8080/helloworld

| Application loaded in interactive mode. Type 'exit' to shutdown.

| Downloading: plugins-list.xml

More information on the [run-app](http://grails.org/doc/latest/ref/Command%20Line/run-app.html) command can be found in the reference guide of grails web site.

# 9. Testing an Application

The create-\* commands in Grails automatically create unit or integration tests for you within the test/unit or test/integration directory. It is of course up to you to populate these tests with valid test logic, information on which can be found in the section on [Testing](http://grails.org/doc/latest/guide/single.html#testing).

To execute tests you run the [test-app](http://grails.org/doc/latest/ref/Command%20Line/test-app.html) command as follows:

grails test-app

# 10. Deploying an Application

Grails applications are deployed as Web Application Archives (WAR files), and Grails includes the war command for performing this task:

grails war

This will produce a WAR file under the target directory which can then be deployed as per your container's instructions.

Unlike most scripts which default to the development environment unless overridden, the war command runs in the production environment by default. You can override this like any script by specifying the environment name, for example:

grails dev war

NEVER deploy Grails using the [run-app](http://grails.org/doc/latest/ref/Command%20Line/run-app.html) command as this command sets Grails up for auto-reloading at runtime which has a severe performance and scalability implications

When deploying Grails you should always run your containers JVM with the -server option and with sufficient memory allocation. A good set of VM flags would be:

-server -Xmx512M -XX:MaxPermSize=256m

Hope you will enjoy working with grails.

I will write more documents for working with grails. Keep your eyes open for new docs.

# 11. More detail domain based project creation

## 11.1 Create application

### 11.1.1 Create new project

Create application using the following commands

grails create-app FinanceManagementSystem

### 11.1.2 Create domain class

Create domain class by using the following command

grails create-domain-class com.progoti.User

## 11.2 Create project in IntelliJ Idea

### 11.2.1 Open IntelliJ Idea

Choose File->New Project

Choose Create project from existing source

Click next

Browse and show the newly created project

Click Next

Click Next without changing anything

Add a JSDK by clicking on + button

Press next

Create a grails SDK or choose from existing

Press Next

Finish

#### 11.2.2 Add Attributes and constraints on the domain User

package com.progoti

class User {

Integer id;

String name;

String userName;

String password;

boolean active;

static mapping = {

table("security\_user")

}

static constraints = {

userName(blank:false, size:3..100,unique:true)

password(blank:false, size:4..100)

}

public static void initialize(){

}

}

#### 11.2.3 Configure database

Add the following to your grails-app\conf\ DataSource.groovy file with the following content make sure to create the dataset in your database server:

dataSource {

pooled = true

url = "jdbc:mysql://localhost/fms?useUnicode=true&characterEncoding=utf8"

driverClassName = "com.mysql.jdbc.Driver"

username = "root"

password = ""

dialect = org.hibernate.dialect.MySQL5UTF8InnoDBDialect

}

hibernate {

cache.use\_second\_level\_cache = false

cache.use\_query\_cache = true

cache.provider\_class = 'net.sf.ehcache.hibernate.EhCacheProvider'

}

// environment specific settings

environments {

development {

dataSource {

dbCreate = "update"

url = "jdbc:mysql://localhost/fms?useUnicode=true&characterEncoding=utf8"

pooled = true

properties {

maxActive = -1

minEvictableIdleTimeMillis=1800000

timeBetweenEvictionRunsMillis=1800000

numTestsPerEvictionRun=3

testOnBorrow=true

testWhileIdle=true

testOnReturn=true

validationQuery="SELECT 1"

}

}

}

test {

dataSource {

dbCreate = "update"

url = "jdbc:mysql://localhost/fms?useUnicode=true&characterEncoding=utf8"

pooled = true

properties {

maxActive = -1

minEvictableIdleTimeMillis=1800000

timeBetweenEvictionRunsMillis=1800000

numTestsPerEvictionRun=3

testOnBorrow=true

testWhileIdle=true

testOnReturn=true

validationQuery="SELECT 1"

}

}

}

production {

dataSource {

dbCreate = "update"

url = "jdbc:mysql://localhost/fms?useUnicode=true&characterEncoding=utf8"

properties {

maxActive = -1

minEvictableIdleTimeMillis=1800000

timeBetweenEvictionRunsMillis=1800000

numTestsPerEvictionRun=3

testOnBorrow=true

testWhileIdle=true

testOnReturn=true

validationQuery="SELECT 1"

}

}

}

}

#### 11.2.4 Configure database driver dependency

Uncomment // runtime 'mysql:mysql-connector-java:5.1.20' line from grails-app\conf\ buildConfig.groovy

You may copy paste the following content to your buildConfig.groovy

grails.servlet.version = "2.5" // Change depending on target container compliance (2.5 or 3.0)

grails.project.class.dir = "target/classes"

grails.project.test.class.dir = "target/test-classes"

grails.project.test.reports.dir = "target/test-reports"

grails.project.target.level = 1.6

grails.project.source.level = 1.6

//grails.project.war.file = "target/${appName}-${appVersion}.war"

grails.project.dependency.resolution = {

// inherit Grails' default dependencies

inherits("global") {

// specify dependency exclusions here; for example, uncomment this to disable ehcache:

// excludes 'ehcache'

}

log "error" // log level of Ivy resolver, either 'error', 'warn', 'info', 'debug' or 'verbose'

checksums true // Whether to verify checksums on resolve

repositories {

inherits true // Whether to inherit repository definitions from plugins

grailsPlugins()

grailsHome()

grailsCentral()

mavenLocal()

mavenCentral()

// uncomment these (or add new ones) to enable remote dependency resolution from public Maven repositories

//mavenRepo "http://snapshots.repository.codehaus.org"

//mavenRepo "http://repository.codehaus.org"

//mavenRepo "http://download.java.net/maven/2/"

//mavenRepo "http://repository.jboss.com/maven2/"

}

dependencies {

// specify dependencies here under either 'build', 'compile', 'runtime', 'test' or 'provided' scopes eg.

runtime 'mysql:mysql-connector-java:5.1.20'

}

plugins {

runtime ":hibernate:$grailsVersion"

runtime ":jquery:1.7.2"

runtime ":resources:1.1.6"

// Uncomment these (or add new ones) to enable additional resources capabilities

//runtime ":zipped-resources:1.0"

//runtime ":cached-resources:1.0"

//runtime ":yui-minify-resources:0.1.4"

build ":tomcat:$grailsVersion"

runtime ":database-migration:1.1"

compile ':cache:1.0.0'

}

}

#### 11.2.5 Generate all the related controller and gsp (view) pages for User Domain

Generate all the necessary controller and views by using the following command:

grails generate-all com.progoti.User

#### 11.2.6 Run the application

Execute the development application by using the following command:

Grails –Dserver.port=8088 run-app

# 12 Web Services

Web services are all about providing a web API onto your web application and are typically implemented in either [REST](http://en.wikipedia.org/wiki/Representational_State_Transfer)

## 12.1 REST

REST is not really a technology in itself, but more an architectural pattern. REST is very simple and just involves using plain XML or JSON as a communication medium, combined with URL patterns that are "representational" of the underlying system, and HTTP methods such as GET, PUT, POST and DELETE.

Each HTTP method maps to an action type. For example GET for retrieving data, PUT for creating data, POST for updating and so on. In this sense REST fits quite well with [CRUD](http://grails.org/doc/latest/guide/single.html#scaffolding).

#### URL patterns

The first step to implementing REST with Grails is to provide RESTful [URL mappings](http://grails.org/doc/latest/guide/single.html#urlmappings):

static mappings = {

"/product/$id?"(resource:"product")

}

This maps the URI /product onto a ProductController. Each HTTP method such as GET, PUT, POST and DELETE map to unique actions within the controller as outlined by the table below:

|  |  |
| --- | --- |
| **Method** | **Action** |
| GET | show |
| PUT | update |
| POST | save |
| DELETE | delete |

In addition, Grails provides automatic XML or JSON marshalling for you.

You can alter how HTTP methods are handled by using URL Mappings to [map to HTTP methods](http://grails.org/doc/latest/guide/single.html#mappingHTTP):

"/product/$id"(controller: "product") {

action = [GET: "show", PUT: "update", DELETE: "delete", POST: "save"]

}

However, unlike the resource argument used previously, in this case Grails will not provide automatic XML or JSON marshalling unless you specify the parseRequest argument:

"/product/$id"(controller: "product", parseRequest: true) {

action = [GET: "show", PUT: "update", DELETE: "delete", POST: "save"]

}

#### HTTP Methods

In the previous section you saw how you can easily define URL mappings that map specific HTTP methods onto specific controller actions. Writing a REST client that then sends a specific HTTP method is then easy (example in Groovy's HTTPBuilder module):

import groovyx.net.http.\*

import static groovyx.net.http.ContentType.JSON

def http = new HTTPBuilder("http://localhost:8080/amazon")

http.request(Method.GET, JSON) {

url.path = '/book/list'

response.success = { resp, json ->

for (book in json.books) {

println book.title

}

}

}

Issuing a request with a method other than GET or POST from a regular browser is not possible without some help from Grails. When defining a [form](http://grails.org/doc/latest/ref/Tags/form.html) you can specify an alternative method such as DELETE:

<g:form controller="book" method="DELETE">

..

</g:form>

Grails will send a hidden parameter called \_method, which will be used as the request's HTTP method. Another alternative for changing the method for non-browser clients is to use the X-HTTP-Method-Override to specify the alternative method name.

#### XML Marshalling - Reading

The controller can use Grails' [XML marshalling](http://grails.org/doc/latest/guide/single.html#xmlAndJSON) support to implement the GET method:

import grails.converters.XML

class ProductController {

def show() {

if (params.id && Product.exists(params.id)) {

def p = Product.findByName(params.id)

render p as XML

}

else {

def all = Product.list()

render all as XML

}

}

..

}

If there is an id we search for the Product by name and return it, otherwise we return all Products. This way if we go to /products we get all products, otherwise if we go to /product/MacBook we only get a MacBook.

#### XML Marshalling - Updating

To support updates such as PUT and POST you can use the [params](http://grails.org/doc/latest/ref/Controllers/params.html) object which Grails enhances with the ability to read an incoming XML packet. Given an incoming XML packet of:

<?xml version="1.0" encoding="ISO-8859-1"?>

<product>

<name>MacBook</name>

<vendor id="12">

<name>Apple</name>

</vender>

</product>

you can read this XML packet using the same techniques described in the [Data Binding](http://grails.org/doc/latest/guide/single.html#dataBinding) section, using the [params](http://grails.org/doc/latest/ref/Controllers/params.html) object:

def save() {

def p = new Product(params.product)

if (p.save()) {

render p as XML

}

else {

render p.errors

}

}

In this example by indexing into the params object using the product key we can automatically create and bind the XML using the Product constructor. An interesting aspect of the line:

def p = new Product(params.product)

is that it requires no code changes to deal with a form submission that submits form data, or an XML request, or a JSON request.

If you require different responses to different clients (REST, HTML etc.) you can use [content negotation](http://grails.org/doc/latest/guide/single.html#contentNegotiation)

The Product object is then saved and rendered as XML, otherwise an error message is produced using Grails' [validation](http://grails.org/doc/latest/guide/single.html#validation) capabilities in the form:

<error>

<message>The property 'title' of class 'Person' must be specified</message>

</error>

## 12.2 REST with JAX-RS (Jersey)

How to start working with Jersey in grails

The following easy steps will help you to start with Jersey web service( RESTful ) using grails:

1. Install and configure grails. Please make sure to read http://grails.org/doc/latest/guide/gettingStarted.html
2. Open command prompt and type "grails create-app TestApp-WS" to create an application using grails
3. Install jaxrs plug-in using "grails install-plugin jaxrs" , this will download the latest released version of the plugin from the [Grails Plugin Repository](http://www.grails.org/plugin/home). For further installation options, such as installing a development snapshot, refer to the [installation instructions](http://code.google.com/p/grails-jaxrs/wiki/InstallationInstructions).
4. **To create a JAX-RS resource named test enter**

grails create-resource test

This will create a TestResource.groovy file under grails-app/resources and a TestResourceTests.groovy file under test/unit. The TestResourceTests.groovy file is a unit test template. The TestResource.groovy file is the generated JAX-RS resource. Both files are in the testapp.ws package.

package testapp.ws  
  
import javax.ws.rs.GET  
import javax.ws.rs.Path  
import javax.ws.rs.Produces  
  
  
@Path('/api/test')  
class TestResource {  
  
    @GET  
    @Produces('text/plain')  
    String getTestRepresentation() {  
        'Test'  
    }  
      
}

It defines a single method that responds to HTTP GET operations. The HTTP response contains the return value of this method, Test in this example. The content type of the response (Content-Type header) is text/plain. The created resource is ready to use as shown in the next section.

Creating resources via the command line is only one option. An alternative is to create resource files by hand. Any \*Resource.groovy file created under grails-app/resources is assumed to be a JAX-RS resource and auto-detected by the grails-jaxrs plugin. These resources are checked for the presence of JAX-RS annotations as defined by JAX-RS 1.1 specification, section 3.1. Resources that aren't properly annotated are ignored by the plugin.

1. **Run the application**

To start the application enter

grails run-app

on the comamnd line. Then open a browser window and go to http://localhost:8080/TestApp-WS/api/test. The browser should now display "Test".

1. **Change the code**

The grails-jaxrs plugin also support code changes at runtime i.e. without restarting the server. To demonstrate that we let add a name parameter to the getTestRepresentation method and bind it to a name query parameter using the JAX-RS @QueryParam annotation. The HTTP response entity will vary depending on the name query parameter. Here's the modified source code.

package testapp.ws  
  
import javax.ws.rs.GET  
import javax.ws.rs.Path  
import javax.ws.rs.Produces  
import javax.ws.rs.QueryParam  
  
  
@Path('/api/test')  
class TestResource {  
  
    @GET  
    @Produces('text/plain')  
    String getTestRepresentation(@QueryParam('name') String name) {  
        "Hello ${name ? name : 'unknown'}"  
    }  
      
}

When you save the changes the plugin re-initializes the JAX-RS runtime. Go to http://localhost:8080/TestApp-WS/api/test?name=Afzal and you should see Hello Afzal in the browser window. If you additionally want to factor out the greeting logic into a Grails service, refer to the [service injection](http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#Service_injection) section for instructions.

1. Generate WADL

**Available in version 0.4 or higher**. A WADL document for resources managed by the plugin can be generated by sending a GET request to http://localhost:8080/TestApp-WS/application.wadl. The result should look like

<application xmlns="http://research.sun.com/wadl/2006/10">  
  <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.1.4.1 11/24/2009 01:30 AM"/>  
  <resources base="http://localhost:8080/ TestApp-WS/">  
    <resource path="/api/test">  
      <method name="GET" id="getTestRepresentation">  
        <request>  
          <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="name"/>  
        </request>  
        <response>  
          <representation mediaType="text/plain"/>  
        </response>  
      </method>  
    </resource>  
  </resources>  
</application>

Generating WADL documents only works when the plugin is [configured](http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#JAX-RS_implementation) to use Jersey as JAX-RS implementation.

1. Create a domain class

To create a Person domain class go to the project's root directory and enter

grails create-domain-class person

Open the generated Person.groovy file (under grails-app/domain) and add two properties, firstName and lastName.

package testapp.ws  
  
class Person {  
  
    static constraints = {  
    }  
      
    String firstName  
      
    String lastName  
      
}

1. **Generate the REST API**

To generate JAX-RS resources that implement the RESTful service interface for that domain class enter

grails generate-resources testapp.ws.Person

This will generate two resource classes, PersonCollectionResource.groovy and PersonResource.groovy (in the testapp.ws package) that support HTTP POST, GET, PUT and DELETE operations for creating, reading, updating and deleting Person objects, respectively. PersonCollectionResource.groovy is related to Person lists, PersonResource.groovy is related to individual Person instances. Let's take a look at how to use the generated RESTful service interface.

1. **Use the REST API**

Start the TestApp-WS application with

grails run-app

New person objects can be created by POSTing to http://localhost:8080/TestApp-WS/api/person. The following request POSTs an XML representation of a person object.

POST /TestApp-WS/api/person HTTP/1.1  
Content-Type: application/xml  
Accept: application/xml  
Host: localhost:8080  
Content-Length: 82  
  
<person>  
  <firstName>Afzalur</firstName>  
  <lastName>Rashid</lastName>  
</person>

The Content-Type header must be set either to application/xml. After sending the request, the server creates a new person object in the database and returns an XML representation of it.

HTTP/1.1 201 Created  
Content-Type: application/xml  
Location: http://localhost:8080/TestApp-WS/api/person/1  
Transfer-Encoding: chunked  
Server: Jetty(6.1.14)  
  
<?xml version="1.0" encoding="UTF-8"?>  
<person id="1">  
  <firstName>Afzalur</firstName>  
  <lastName>Rashid</lastName>  
</person>

The client explicitly requested an XML representation via the Accept request header. Note that the returned representation differs from the submitted representation by an id attribute in the <person> element. This id is also contained in the Location response header, the URL of the created resource. The response code is 201 (CREATED). Let's create another person object using a JSON representation. Here's the request

POST /TestApp-WS/api/person HTTP/1.1  
Content-Type: application/json  
Accept: application/json  
Host: localhost:8080  
Content-Length: 58  
  
{"class":"Person","firstName":"Fabien","lastName":"Barel"}

The response also contains a JSON representation of the created person (see Accept request header). The id of the created person object is 2.

HTTP/1.1 201 Created  
Content-Type: application/json  
Location: http://localhost:8080/TestApp-WS/api/person/2  
Transfer-Encoding: chunked  
Server: Jetty(6.1.14)  
  
{"class":"Person","id":"2","firstName":"Fabien","lastName":"Barel"}

Content negotiation via Content-Type and Accept headers works for other HTTP methods as well. To GET a list of created persons, open a browser (Firefox in our example) and enter the URL http://localhost:8080/TestApp-WS/api/person. This returns an XML representation of the list of persons stored in the database.

To learn more about this please visit

http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures

http://code.google.com/p/grails-jaxrs/wiki/AdvancedFeatures#Using\_GORM

http://jersey.java.net/use/getting-started.html

http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html