

About Grails

Grails is a full stack framework and attempts to solve as many pieces of the web development puzzle through the core technology and it's associated plug-ins. Included out the box are things like:

- An easy to use Object Relational Mapping (ORM) layer built on [Hibernate](#)
- An expressive view technology called Groovy Server Pages (GSP)
- A controller layer built on [Spring MVC](#)
- A command line scripting environment built on the Groovy-powered [Gant](#)
- An embedded Jetty container which is configured for on the fly reloading
- Dependency injection with the inbuilt [Spring](#) container
- Support for internationalization (i18n) built on Spring's core MessageSource concept
- A transactional service layer built on Spring's transaction abstraction

All of these are made easy to use through the power of the [Groovy](#) language and the extensive use of Domain Specific Languages (DSLs).

Website of Grails : www.grails.org

Model / Domain

A Domain fulfills the M in the Model View Controller (MVC) pattern and represents a persistent entity that is mapped onto an underlying database table. In Grails a domain is a class that lives in the grails-app/domain directory. A domain class can be created with the create-domain-class command:

```
grails create-domain-class Book
```

Example of Domain class:

```
class Book {
    String title
    Date releaseDate
    Author author
}
```

The class name, by default, is mapped to the table name in lower case and separated by underscores instead of camel case. While each property maps to individual columns.

GORM

GORM is Grails' object relational mapping (ORM) implementation. Under the hood it uses Hibernate 3 (an extremely popular and flexible open source ORM solution) but because of the dynamic nature of Groovy, the fact that it supports both static and dynamic typing, and the convention of Grails there is less configuration involved in creating Grails domain classes.

One to One

Unidirectional

```
class Face {
    Nose nose
}

class Nose {
}
```

Unidirectional with cascade

```
class Face {
    Nose nose
}

class Nose {
    static belongsTo = Face
}
```

Bidirectional relationship – no cascade

```
class Face {
    Nose nose
}

class Nose {
    Face face
}
```

Bidirectional with cascade

```
class Face {
    Nose nose
}

class Nose {
    static belongsTo = [face:Face]
}
```

One to Many

Grails will, by default, map this kind of relationship with a join table. Grails will automatically inject a property of type `java.util.Set` into the domain class based on the `hasMany` setting.

No cascade delete

```
class Author {
    static hasMany = [ books : Book ]
    String name
}

class Book {
    String title
}
```

With cascade delete

```
class Author {
    static hasMany = [ books : Book ]
    String name
}

class Book {
    static belongsTo = [author:Author]
    String title
}
```

MappedBy for multiple One to Many

```
class Airport {
  static hasMany = [outboundFlights:Flight,
  inboundFlights:Flight]
  static mappedBy =
  [outboundFlights:"departureAirport",
  inboundFlights:"destinationAirport"]
}

class Flight {
  Airport departureAirport
  Airport destinationAirport
}
```

Many to Many

Grails maps a many-to-many using a join table at the database level.

```
class Book {
  static belongsTo = Author
  static hasMany = [authors:Author]
  String title
}

class Author {
  static hasMany = [books:Book]
  String name
}
```

The owning side of the relationship, in this case `Author`, takes responsibility for persisting the relationship and is the only side that can cascade saves across.

Controller

A controller fulfills the C in the Model View Controller (MVC) pattern and is responsible for handling web requests. In Grails a controller is a class that ends in the convention "Controller" and lives in the `grails-app/controllers` directory. A controller can be created with the `create-controller` command:

```
grails create-controller hello
```

Example of Controller:

```
class HelloController {
  def world = {
    render "Hello World!"
  }
}
```

Each action in the controller is a callable block (for example the `world` action above) that has access to a number of implicit variables and methods.

View

Groovy Servers Pages (or GSP for short) is Grails' view technology. It is designed to be familiar for users of technologies such as ASP/JSP, but to be far more flexible and intuitive. In Grails GSPs live in the `grails-app/views` directory and are typically rendered automatically (by

convention) or via the `render` method in controller such as: `render (view:"index")`

Example of GSP:

```
<html>
  <body>
    Hello ${params.name}<br/>
    <% [1,2,3,4].each { num -> %>
      <p><%= "Hello ${num}!" %></p>
    <%}%>
  </body>
</html>
```

Tag Library

A tag library fulfills role of "view helper" in the Model View Controller (MVC) pattern and is responsible aiding GSP rendering. In Grails a tag library is a class that ends in the convention "TagLib" and lives in the `grails-app/taglib` directory.

Example of Grails native Tag

```
<g:set var="now" value="${new Date()}"
scope="request" />
```

Service

A service contains business logic that can be re-used across a Grails application. In Grails a service is a class that ends in the convention "Service" and lives in the `grails-app/services` directory. A service can be created with the `create-service` command:

```
grails create-service Book
```

Example of Service:

```
class BookService {
  Book[] getBooks() {
    Book.list() as Book[]
  }
}
```

Example of using Service in Controller

```
class BookController{
  def bookService

  def index = {
    def books = bookService.getBooks()
  }
}
```

Command

Grails incorporates the powerful build system Gant, which is a Groovy wrapper around Apache Ant. The basic usage scenario is:

```
grails [environment]* [command name]
```

To get a list and some help about the available commands type: `grails help`

Example of Grails command:

```
grails dev run-app
```

bootstrap	The bootstrap command is mainly intended to be used by other scripts and enables the ability to bootstrap a Grails application instance outside of the container for usage in tools that require a reference to Grails' ApplicationContext
bug-report	The bug-report command will package up only the sources of your application (excluding jars, static resources etc.) into a zip file with a timestamp appended. This is useful for reporting issues to the Grails JIRA installation
clean	The clean command will delete all compiled resources from the current Grails application. Since Groovy is a compiled language, as with Java, this is sometimes useful to clear old instances of classes out and ensure correct compilation
compile	The compile command will execute the compile phase of the Grails pre-packaging process, which pre-compiles Groovy and Java sources.
console	Starts an instance of the Swing graphical Groovy console with an initialized Grails runtime.
create-app	The starting point for Grails. This command creates a Grails application and requires the user to specify the application name. A subdirectory within the directory the command was executed from is then created based on the entered application name.
create-controller	The create-controller command will create a controller and associated integration test for the given base name.
create-domain-class	The create-domain-class command will create a domain and associated integration test for the given base name.
create-integration-test	The create-integration-test command will create an integration test for the given base name.
create-plugin	The create-plugin command will create a Grails plug-in project.
create-script	The create-script command will create a new Grails executable script that can be run with the grails command from a terminal window.
create-service	The create-service command will create a Grails service class for the given base name.
create-tag-lib	The create-tag-lib command will create a tag library and associated integration test for the given base name.
create-unit-test	The create-unit-test command will create an unit test for the given base name.
doc	Generates javadoc and groovydoc documentation from the current Grails project.
generate-all	Generates a controller and views for the given domain class
generate-controller	Generates a controller for the given domain class
generate-views	Generates a set if views for the given domain class
help	Displays Grails command line help
init	The init command initialises the appropriate directory structure for a Grails application and set some common variables that can be used by plug-ins and other scripts. It is typically used as an include.
install-plugin	Installs a plug-in from a file, URL or the Grails central SVN repository
install-templates	Installs the templates used by Grails during code generation
list-plugins	Lists the plug-ins available from the Grails standard repository
package-plugin	Packages a plug-in as a zip archive which

	can then be installed into another application
package	Runs the packaging phase of Grails' runtime. This is mainly useful when used by other scripts.
plugin-info	Displays detailed info about a given plug-in
release-plugin	Tags and releases a Grails plug-in to the Grails plug-in repository
run-app-https	Extended version of run-app that runs Grails using a generated keystore and SSL certificate for HTTPS
run-app	Runs Grails uses an embedded Jetty container on port 8080
run-war	Packages the current Grails application into a Web Application Archive (WAR) file and runs the application in a Jetty container on port 8080
set-proxy	Sets the proxy Grails should use when communicating over the internet such as with the install-plugin command
set-version	Sets the current application version inside the application metadata file (application.properties)
shell	Starts an instance of the Groovy terminal shell with an initialized Grails runtime.
stats	Outputs basic statistics about the current Grails application, including number of files, line count and so on
test-app	Runs all Grails unit and integration tests and outputs reports. The command will return appropriate response codes for embedding with continuous integration servers.
upgrade	The upgrade command will upgrade the current Grails application to currently installed version of Grails if possible.
war	The war command will create a Web Application Archive (WAR) file which can be deployed on any Java EE compliant application server

Model Reference

* means need to combine with the fields or domains,

Example: addTo*, the usage is

```
author.addToBooks(book)
```

addTo*	Adds a domain class relationship for one-to-many or many-to-many relationship, where the relationship is indicated by the property used as the suffix to the method.
belongsTo	Defines a "belongs to" relationship where the class specified by belongsTo assumes ownership of the relationship. This has the effect of controlling how deletes cascade. In other words the owning side will cascade deletes when belongsTo is specified on the inverse side.
clearErrors	Clear the list of errors on a domain class. This may be useful if a domain class contains errors because of binding problems or validation problems. The errors may be corrected programatically. Validation may continue to fail unless the errors are cleared.
constraints	Allows the definition of declarative validation constraints.
count	Counts the number of instances in the database and returns the result
countBy*	Dynamic method that uses the properties of the domain class to allow the creation of Grails query method expressions that count the number of records returned
createCriteria	Creates an returns an instance of Grails' HibernateCriteriaBuilder that can be used to construct criteria queries
delete	Indicates that a persistent instance should be deleted.
discard	Discards any changes that have been made

	during an update.
errors	An instance of the Spring Errors interface that contains data binding and/or validation errors.
executeQuery	Allows the execution of HQL queries against a domain class
executeUpdate	Allows updating a database with DML-style operation
exists	Checks whether an instance exists for the specified id and returns true if it does
fetchMode	Allows the configuration of an associations fetch strategy ('eager' or 'lazy')
find	Finds and returns the first result for the given query or null if no instance was found
findAll	Finds all of the domain class instances for the specified query
findAllBy*	Dynamic method that uses the properties of the domain class to allow the creation of Grails query method expressions that return all instances of the domain class
findAllWhere	Uses named arguments that match the property names of the domain class to produce a query that returns all of the matching results.
findBy*	Dynamic method that uses the properties of the domain class to allow the creation of Grails query method expressions that return the first result of the query
findWhere	Uses named arguments that match the property names of the domain class to produce a query that returns the first result.
get	Retrieves an instance of the domain class for the specified id, otherwise returns null
getAll	Retrieves an instances of the domain class for the specified ids and returns a list of these instances, ordered by the original ids list. If some of provided ids are null or there are no instances with these ids, then result list will contain null values on corresponding positions.
hasErrors	True if the domain class instance has errors following a call to validate or save or following Data Binding
hasMany	Defines a one-to-many association between two classes.
ident	Returns the value of the identity property of the domain class regardless of the name of the identity property itself
list	Lists all of the instances of the domain class.
listOrderBy*	Lists all of the instances of the domain class ordered by the property in the method expression
lock	The lock method obtains a pessimistic lock using an SQL select ... for update.
mappedBy	The mappedBy static property adds the ability to change how a bidirectional associations is linked
mapping	The mapping static property is used to allow how GORM maps to the underlying database.
merge	Merges a domain class instance back into the current persistent context and returns a new merged instance.
properties	Allows access to the domain class properties as a map and is typically used for Data Binding to perform types conversion when set allowing properties to be set from request parameters.
refresh	Refreshes a domain classes state from the database
removeFrom*	Opposite of the addTo method in that it removes instances from an association.
save	Saves a domain class instance to the database cascading updates to any child instances if required.
transients	Defines a list of property names that should not be persisted to the database. This is often useful if you have read-only getters that include logic.
validate	Validates a domain class against the applied constraints
withCriteria	Allows inline execution of criteria with a closure.
withTransaction	Allows programmatic transactions using Spring's Transaction Abstraction and a block.

actionName	Returns the name of the currently executing action which is dictated by the URL mappings
afterInterceptor	To define an interceptor that is executed after an action use the afterInterceptor property:
allowedMethods	To limit access to controller actions based on the HTTP request method.
beforeInterceptor	Allows the interception of an action before it is executed. A beforeInterceptor can optionally halt execution of the action all together.
bindData	Allows fine grained control of binding request parameters from strings onto objects and the necessary types (data binding)
chain	Uses flash storage to implicitly retain the model following an HTTP redirect from one action to another.
controllerName	Returns the name of the currently executing controller
flash	A temporary storage map that stores objects within the session for the next request and the next request only, automatically clearing our the objects held there after the next request completes.
grailsApplication	The GrailsApplication class provides information about the conventions within Grails and access to metadata, config and the ClassLoader
params	A mutable multi-dimensional map (hash) of request (CGI) parameters
redirect	To redirect flow from one action to the next using an HTTP redirect.
render	To render different forms of responses from simple text responses, to view and templates.
request	The request object is an instance of the Servlet API's HttpServletRequest class
response	The response object is an instance of the Servlet API's HttpServletResponse class
ServletContext	The ServletContext object is an instance of the Servlet API's ServletContext class.
session	The session object is an instance of the Servlet API's HttpSession class
withFormat	Used to execute different responses based on the incoming request Accept header, format parameter or URI extension.

Domain Constraint

Constraints provide Grails with a declarative DSL for defining validation rules, schema generation and CRUD generation meta data. An example of a set of constraints applied to a domain class are as follows:

```
class User {
    static constraints = {
        login(size:5..15, blank:false, unique:true)
        password(size:5..15, blank:false)
        email(email:true, blank:false)
    }
}
```

Name	Function
blank	To validate that a String value is not blank
creditCard	To validate that a String value is a valid credit card number
email	To validate that a String value is a valid email address.
inList	To validate that a value is within a range or collection of constrained values.
matches	To validate that a String value matches a given regular expression.
max	Ensures a value does not exceed the given maximum value.
maxSize	Ensures a value's size does not exceed the given maximum value.
min	Ensures a value does not fall below the given minimum value.
minSize	Ensures a value's size does not fall below the given minimum value.
notEqual	Ensures that a property is not equal to the specified

Controller Reference

	value
nullable	Ensures the property value cannot be null when set to false
range	Uses a Groovy range to ensure that a property's value occurs within a specified range
scale	Set to the desired scale for floating point numbers (i.e., the number of digits to the right of the decimal point).
size	Uses a Groovy range to restrict the size of a collection or number or the length of a String.
unique	Constraints a property as unique at the database level
url	To validate that a String value is a valid URL.
validator	Adds custom validation to a field.

	regex patterns etc.
hasErrors	Checks whether a bean, request scope, or model reference has any errors and if it does invokes the body of the tag. Typically used in conjunction with either eachError or renderErrors
Header	Obtains the value of a named header
hiddenField	Creates a input of type 'hidden' (a hidden field). All the usual HTML elements apply.
if	The logical if tag to switch on an expression and/or current environment.
javascript	Allows inclusion of javascript libraries and scripts as well a shorthand for inline Javascript
layoutBody	Used in layouts to output the contents of the body tag of the decorated page.
layoutHead	Used in layouts to output the contents of the head tag of the decorated page. Equivalent to the SiteMesh <decorator:head /> tag.
layoutTitle	Used in layouts to output the contents of the title tag of the decorated page. Equivalent to the SiteMesh <decorator:title /> tag.
link	Creates an html anchor tag with the href set based on the parameters specified.
localeSelect	Creates a select to select from a list of available locales
message	Resolves a message from the given code or error. Normally used in conjunction with eachError
meta	Renders application metadata properties.
pageProperty	Used in layouts to output the contents a property of the decorated page. Equivalent to the SiteMesh <decorator:getProperty/> tag.
paginate	Creates next/previous buttons and a breadcrumb trail to allow pagination of results
passwordField	Creates a input of type 'password' (a password field). An implicit "id" attribute is given the same value as name unless you explicitly specify one.
radio	Helper tag for a radio button
radioGroup	Helper tag for creating radio button groups
remoteField	Creates a text field that sends its value to a remote link when it changes. By default the parameter name sent is called 'value', this can be changed by specifying a 'paramName' attribute.
remoteFunction	Creates a remote javascript function that can be assigned to a DOM event to call the remote method
remoteLink	Creates a link that calls a remote function when clicked
render	Applies an inbuilt or user defined Groovy template against a model so that templates can be re-used for lists or instance or a single instance
renderErrors	Allows rendering of errors in different formats (at the moment only an HTML list is implemented)
select	Helper tag for creating HTML selects.
set	Set the value of a variable accessible with the GSP page.
sortableColumn	Renders a sortable column to support sorting in tables.
submitButton	Creates a submit button with the indicated value. Javascript event handlers can be added using the same parameter names as in HTML.
submitToRemote	Creates a button that submits the surrounding form as a remote ajax call serializing the fields into parameters.
textArea	Creates a HTML text area element. An implicit "id" attribute is given the same value as name unless you explicitly specify one.
textField	Creates a input of type 'text' (a text field). An implicit "id" attribute is given the same value as name unless you explicitly specify one.
timeZoneSelect	Helper tag for creating HTML selects for selecting from a list of time zones
uploadForm	Creates a form that can submit multi-part data used in file uploads.
while	Executes a condition in a loop until the condition returns false

TagLibs Reference

Usage example: <g:submitButton>

TagName	Description
actionSubmit	Creates a submit button with the indicated value. Javascript event handlers can be added using the same parameter names as in HTML.
applyLayout	Applies the specified layout to either the body, a given template and an arbitrary URL allowing the development of "portlet" style applications and mashups
checkBox	Creates a checkbox form field. All the usual HTML elements apply.
collect	Uses the Groovy JDK collect method to iterate over each element of the specified object transforming the result using the expression in the closure
cookie	Obtains the value of a named cookie
createLink	Creates a link that can be used where necessary (for example in an href, javascript, ajax call etc.)
currencySelect	Helper tag for creating HTML selects for selecting from a list of currency symbols (eg. 'EUR', 'USD' etc.).
datePicker	Creates a date picker which renders as selects for the day,month,year,hour and second of the day.
each	Uses the Groovy JDK each method to iterate over each element of the specified object.
eachError	Loops through each error of the specified bean or model. If no arguments are specified it will go through all model attributes and check for errors.
else	The logical else tag
elseif	The logical elseif tag
fieldValue	This tag will inspect a bean which has been the subject of data binding and obtain the value of the field either from the originally submitted value contained within the bean's errors object populating during data binding or from the value of a bean's property. Once the value is obtained it will be automatically HTML encoded.
findAll	Uses the Groovy JDK findAll method to iterate over each element of the specified object that match the GPath expression within the attribute "expr"
form	Creates a form that submits to a controller, action, and/or id. Beyond what is below all the usual HTML attributes apply.
formRemote	Creates a form tag that uses a remote uri to execute an ajax call serializing the form elements falling back to a normal form submit if javascript is not supported.
formatDate	Allows the formatting of java.util.Date instances using the same patterns defined by the SimpleDateFormat class.
formatNumber	Allows the formatting of numbers using the same patterns defined by the DecimalFormat class.
grep	Uses the Groovy JDK grep method to iterate over each element of the specified object that match the specified "filter" attribute. The filter can be different instances such as classes,