Extracted from:

Grails 2: A Quick-Start Guide

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Grails 2
A Quick-Start Guide

Dave Klein
Ben Klein

GRAILS

Grails 2: A Quick-Start Guide

Dave Klein
Ben Klein

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Susannah Davidson Pfalzer (project manager)
Potomac Indexing, LLC (indexer)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-77-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—December 2013

Tell me and I forget. Teach me and I remember.
Involve me and I learn.

➤ *Benjamin Franklin*

CHAPTER 2

Our Project

When you're learning a new tool or language, you might start with a "Hello World" example or perhaps work through a few exercises in a book. Those steps can help you become acquainted with the tool, but that's as far as they'll take you. If you want to become productive in a tool or even proficient, you need use it in a real project. So, that's what we're going to do. We'll work together to build a cool new web application—one that will actually go live. As our application comes together, we'll explore Grails in a thorough, practical way. This strategy will provide us with the context that is so valuable in understanding and becoming productive with a new framework.

We'll be working through a series of iterations, covering about one iteration per chapter. This means that some features of Grails will be used in more than one chapter. We want to build a real application, and the repetition that comes with that is a good thing. This is a quick-start guide, but we don't want it to be a false-start guide. When our time together is over, you'll be able to go on to your second Grails project with confidence.

One concern with this method of discovery is that we're going to run into more advanced features of Grails, perhaps before we are ready. We'll handle this potential problem by developing our application in an incremental manner. In other words, our application will start simple, thereby exercising the simple features in Grails, and gradually get more complex.

Introducing TekDays.com

The decision about what kind of project to take on in our quest to learn Grails is an important one. We want something that is substantial enough to exercise the framework in ways that will stick in our minds but not something that is so daunting that we are unable to finish it. We're also aiming for something useful *and* interesting. After all, you may need something more than our charm and wit to keep your attention.

Here's an issue many developers encounter: the rapid pace of technological innovation today is making it more difficult and, at the same time, increasingly important to keep our skills as developers up-to-date. One great way to keep on top of innovations and advances is to attend technical conferences, but with tightening training budgets at many companies and more developers working as freelancers or independent contractors, it is often hard to afford these events. Some developers have taken to organizing local, nonprofit mini-conferences to help address the problem. You may have heard of these events, such as the Houston Tech Fest, Silicon Valley Code Camp, or the bar camps that are springing up all over.[1] Wouldn't it be great if there was an online application to help individuals connect and put on these types of events? Well, when we're done here, there will be!

TekDays.com is going to be a site where people can announce, plan, and promote local, grassroots technical conferences. It will all start when visionary individuals suggest an event in their city. Then, as others hear about it and register their interest and/or support, we'll provide tools to help them organize the event: a to-do list, an organizer's dashboard (to keep track of volunteers, sponsors, and potential attendees), a discussion forum, and, finally, an event home page to help with promotion. This may sound like a tall order, but Grails can make it happen.

Meet Our Customer

One of the major benefits of Grails is its ability to provide rapid feedback. In minutes, we can have new features up and running and ready for our customers to try. But that benefit is hard to realize if we don't have a customer around. And this application is about building community: making connections, sharing ideas, and working together to build a solution. This application is going to production; in fact, we're going to use it to organize a real tech conference, so your authors, Dave and Ben, will be joining you on the dev team as well as playing the role of on-site customer—*and* first end user. Don't worry; we have experience wearing multiple hats. As we work on TekDays, you can show us what you've done, and we'll let you know what we think about it. Fair enough?

1. For more information on these events, see http://www.houstontechfest.org, http://www.siliconvalley-codecamp.com, and http://en.wikipedia.org/wiki/BarCamp.

Application Requirements

As your *customer*, we want to give you a good idea of what we are looking for in this application. We are trying to attract conference organizers to this site —preferably many of them. We're convinced of the value of these types of conferences to individual developers, communities, and the industry as a whole. The application should make it easy for those visionary individuals to get started by simply proposing a conference. Then it has to provide real help in bringing their vision to fruition.

As *end users*, we're hoping to use this application to organize a technical conference in St. Louis, Missouri. This is a big undertaking, and we know that we can't do it alone, so we need this application to make it easy for others to volunteer, or to at least let us know they're interested in attending. Some type of workflow to guide us through the process would make this whole endeavor much less daunting.

After this introduction and a follow-up discussion with our customer and user, we've come up with the following feature list for our application:

- Create new events
- Display event details
- Edit event details
- Create users/organizers
- Allow users to volunteer to help
- Add users to events
- Allow anonymous users to register interest
- Create sponsors
- Add sponsors to events
- Have default list of tasks
- Add/remove tasks
- Assign tasks to users
- Post forum message
- Reply to forum message
- Display forum message threads
- Allow access to event home page with simple URL

This list gives us a good idea of the scope of the project. When we're done here, people will be able to propose conferences, volunteer to help, or add their support. Organizers will be able to assign tasks to volunteers to spread the load, and questions can be asked and answered in the forums to keep the communication flowing. As a conference begins to take shape, we'll provide the tools needed to promote it successfully. Businesses will be able to bring their resources to bear to help make it all happen. This is getting exciting!

We will, of course, need to flesh these out more as we go along. During each iteration, we'll design and implement two or three features. Along the way, we (or our customer) may come up with new features or changes. That's OK. Grails can handle it, and so can we.

Iteration Zero

Before we get started building our application, we'll take a few moments to set the stage.

Installing Grails

First off, let's get Grails installed and set up. There are a few different ways to install Grails, with installers on one end of the spectrum and building the source from GitHub on the other. We'll use that happy middle ground and download the compressed binaries. They are at http://grails.org/download and are made available as zip files. Once we have them, follow these steps:

1. Expand the archive to a directory on your computer.
2. Set your GRAILS_HOME environment variable to this directory.
3. Add GRAILS_HOME/bin to your path.
4. Ensure that you have a JAVA_HOME environment variable pointing to a JDK version 1.6 or higher.

To test our installation, run the following command:

```
$ grails help
```

If this returns something like the following output, then we're good to go:

```
| Environment set to development.....

Usage (optionals marked with *):
grails [environment]* [options]* [target] [arguments]*

Examples:
grails dev run-app
grails create-app books
```

```
Available options:
 -debug-fork                  Whether to debug the forked JVM if using
                              forked mode
 -verbose                     Enable verbose output
 -plain-output                Disables ANSI output
 -refresh-dependencies        Whether to force a resolve of dependencies
                              (skipping any caching)
 -reloading                   Enable the reloading agent
 -stacktrace                  Enable stack traces in output
 -offline                     Indicates that Grails should not connect
                              to any remote servers during processing of
                              the build
 -version                     Current Grails version
 -non-interactive             Whether to allow the command line to
                              request input

Available Targets (type grails help 'target-name' for more info):
grails add-proxy
grails alias
grails bootstrap
grails bug-report
grails clean
...
```

If you don't see this output, verify that your GRAILS_HOME and JAVA_HOME environment variables are valid and that GRAILS_HOME/bin is on your path. You can do this easily with echo:

```
$ echo $GRAILS_HOME
$ echo $JAVA_HOME
$ echo $PATH
```

On Windows, this would be as follows:

```
> echo %GRAILS_HOME%
> echo %JAVA_HOME%
> echo %PATH%
```

Grails Scripts

Grails comes with more than seventy built-in scripts that can be run with the grails command. These scripts are used for creating applications and application artifacts, as well as to run tests or to run the application. We'll learn about many of these as we work on TekDays. If you want to explore the others, you can do that with grails help. As we saw in the previous section, grails help will show you a list of the scripts that come with the framework. To find out more about any one of them, run grails help followed by the name of the script. For example:

```
$ grails help run-app
```

Although we will be using the built-in scripts only to get TekDays ready for production, it's worth noting that other scripts can be used with the grails command; some plugins install new scripts, and it's also possible to write your own scripts for Grails.

Setting Up Our Workspace

In other web frameworks that we've used—especially Java-based frameworks—starting a new project is an ordeal. If you're lucky, there might be a wizard, or perhaps there's a template project you can copy and customize. Even with those aids, getting everything set up and in the right place can be a drag. Grails has a solution to this problem, in the form of a script called create-app. We'll use this script to get TekDays off the ground.

From the directory that will be the parent of our project directory, enter the following command:

```
$ grails create-app TekDays
```

When we run the command, Grails creates a bunch of directories and files for our project. In just a bit, we'll take a closer look at the directories that are created and what they are used for.

The TekDays project is now ready to go. In fact, we can even run it already:

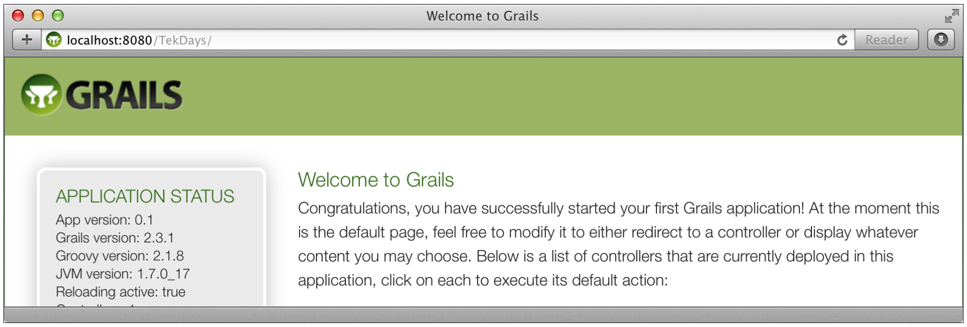
```
$ cd TekDays
$ grails run-app
```

Here's a summarized view of the output from the run-app script:

```
| Loading Grails 2.3.1
| Configuring classpath
| Environment set to development.....
| Running Grails application
| Server running. Browse to http://localhost:8080/TekDays
```

Early on in the output, Grails tells us that the environment is set to development. development is the default of the three standard Grails environments. Running in the development environment (or in development mode, as it is often called) gives us autoreloading (we can change most aspects of the application while it's running and see the changes immediately) and an in-memory database to make that rapid feedback even more rapid. These types of productivity-enhancing features can be added to most other frameworks via external tools and libraries, but Grails bakes them right in. The other two environments are test and production. We'll return to these other environments later when we get to testing and deployment. For now, keep in mind that these are only defaults and can be changed if needed.

The last line of output tells us where to go to see our application in action. In the following figure, we can see what we get by browsing to that location.



APPLICATION STATUS
App version: 0.1
Grails version: 2.3.1
Groovy version: 2.1.8
JVM version: 1.7.0_17
Reloading active: true

Welcome to Grails
Congratulations, you have successfully started your first Grails application! At the moment this is the default page, feel free to modify it to either redirect to a controller or display whatever content you may choose. Below is a list of controllers that are currently deployed in this application, click on each to execute its default action:

Figure 1—We start with a working application.

The default home page of the app displays some application statistics in the sidebar, as well as a list of installed plugins and a list of the app's controllers. (There is only one controller to begin with: grails.plugin.databasemigration.DbdocController is part of the Grails Database Migration plugin,[2] which is automatically installed by Grails.)

It may not look like much yet, but having a working application from the very beginning is just *powerful*. It gives us an excellent feedback loop. We'll be maintaining that runnable state, and, consequently, that feedback loop, right through to deployment.

Starting with All Windows Intact

In their book *The Pragmatic Programmer [HT00]*, Dave Thomas and Andy Hunt discuss the "Broken Window" theory as it relates to software development. This theory holds that if a building has a broken window that is left unrepaired, its chances of further vandalism are increased. Dave and Andy point out that if software is left in a partially broken state (failing tests or ignored bugs), it will continue to degenerate.

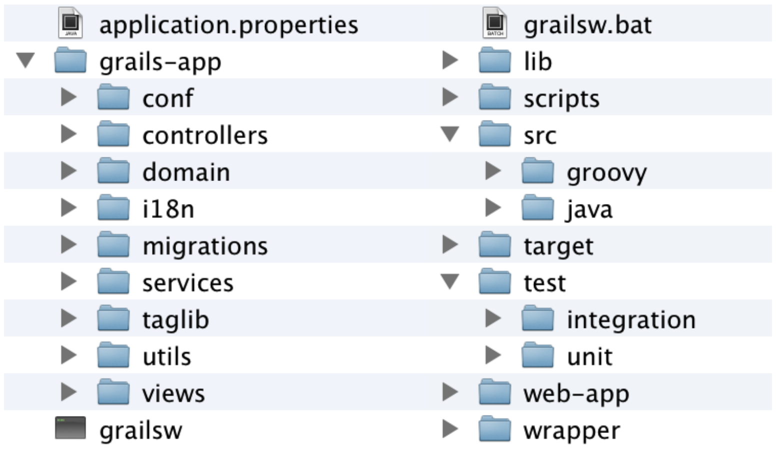
With many development tools and frameworks, we start out with broken windows; nothing works until multiple pieces are in place. This makes it easier to get started and keep coding without taking the time to see whether what we have *works*. With Grails we start out with a running application; as we make changes, we get immediate feedback that lets us know whether we've broken something.

2. See http://grails.org/plugin/database-migration.

With some other web frameworks, we would have had to create one or two source files, an index page, and a handful of XML files to get this far. All it took in Grails was a single command.

Anatomy of a Grails Project

Now that we've seen our application run, let's take a look at what's under the hood. When we ran the create-app script, a number of files and directories were generated for us. (See the next figure.) The files that were created have default code and configuration information that we can change as needed. The directories are particularly important because they are at the heart of Grails' "convention over configuration" strategy. Each directory has a purpose, and when files are placed in these directories and meet certain other conventions, *magical things* will happen. We will look at most of these in more detail when we begin to work with them. For now, here's a brief overview:



```
application.properties          grailsw.bat
▼ grails-app                    ▶ lib
    ▶ conf                      ▶ scripts
    ▶ controllers               ▼ src
    ▶ domain                        ▶ groovy
    ▶ i18n                          ▶ java
    ▶ migrations                ▶ target
    ▶ services                  ▼ test
    ▶ taglib                        ▶ integration
    ▶ utils                         ▶ unit
    ▶ views                     ▶ web-app
grailsw                         ▶ wrapper
```

Figure 2—The files and directories of a Grails application

- grails-app: The main application directory, which contains the following directories:
  - conf: Contains Grails configuration files and directories for optional Hibernate and Spring configuration files[3]
  - controllers: Holds the controller classes, the entry points into a Grails application
  - domain: Holds domain classes, representing persistent data

3. Most Grails applications will not need Spring or Hibernate configuration files.

  - i18n: Holds message property files for internationalization
  - migrations: Can contain change log files generated by the Grails Database Migration plugin
  - services: Holds service classes, which are Spring-managed beans
  - taglib: Holds Groovy Server Pages (GSP) custom tag libraries
  - utils: Holds codec classes[4]
  - views: Holds the GSP views

- lib: Contains any external .jar files we may need to include (such as JDBC drivers).
- scripts: Can contain custom Groovy scripts to be used in the application.
- src: Contains directories for other Java and Groovy source files. Files in this directory are available to the application at runtime.
- target: Created when we first run the app. It contains artifacts produced by Grails commands such as grails war.
- test: Contains directories for unit and integration tests.
- web-app: Contains directories for images, CSS, and JavaScript.
- wrapper: Can contain wrapper files generated by the wrapper script.

The application.properties file holds our application's name and version, along with a list of plugins used. The default version for a new Grails application is 0.1; we can change this in application.properties. The grailsw shell script and grailsw.bat batch file allow our project to be run without a manual installation of Grails; if Grails isn't installed when they're run, they will download it and set it up to work with our project, and can then be used to run Grails scripts in place of the usual grails command.

A brief word about tools: support for Groovy and Grails in most of the popular development tools is good and getting better all the time. Integrated development environments (IDEs) such as Eclipse, NetBeans, and IntelliJ IDEA are a big help in managing a multitude of configuration files or for dealing with verbose and redundant language syntax, but with Grails' use of "convention over configuration" and the clean, concise syntax of Groovy, we find ourselves turning to an IDE less and less. If you really feel the need for an IDE, you can find more information about what's available in Appendix 2, *Resources*, on

4. See http://grails.org/doc/2.3.1/guide/single.html#codecs.

page ?. As we work on TekDays, we'll be using the command line for interacting with Grails, but coding can be done in an editor or IDE.

Summary

We're off to a good start. We have Grails installed. Our project requirements are clear and achievable. Our new application is prepped, ready, and running.

In the next chapter, we'll begin our first development iteration. To get ourselves acclimated, we'll reach for some low-hanging fruit and work on the first three features on our list. At the end of Chapter 3, we will be able to create, display, and edit an event.