Rails from the Outside In

Learning Rails

Simon St.Laurent
& Edd Dumbill

O'REILLY®

Web Programming

O'REILLY®

Learning Rails

While most Rails books cater to programmers looking for information on data structures, *Learning Rails* targets web developers whose programming experience is tied directly to the Web.

Rather than begin with the inner layers of a Rails web application—the models and controllers—this unique book approaches Rails development from the outer layer: the application interface. You'll learn how to create something visible with Rails before reaching the more difficult database models and controller code. With *Learning Rails*, you can start from the foundations of web design you already know, and then move more deeply into Ruby, objects, and database structures. This book will help you:

- Present web content by building an application with a basic view and a simple controller, while learning Ruby along the way
- Build forms and process their results, progressing from the simple to the more complex
- Connect forms to models by setting up a database, and use Rails' ActiveRecord to create code that maps to database structures
- Use Rails scaffolding to build applications from a view-centric perspective
- Add common web application elements such as sessions, cookies, and authentication
- Build applications that combine data from multiple tables
- Create simple but dynamic interfaces with Rails and Ajax

Once you've read *Learning Rails*, you'll be comfortable working with the Rails web framework, and you'll be well on your way to becoming a Rails guru.

"Learning a new web framework (and language) can be difficult, but Simon and Edd take it one step at a time and explain everything you need to know to get started with Rails."

—Gregg Pollack,
RailsEnvy.com

Simon St.Laurent is a senior editor at O'Reilly and a web developer. His books include *Programming Web Services with XML-RPC* (O'Reilly), *XML: A Primer* (Wiley), and *Office 2003 XML* (O'Reilly).

Edd Dumbill is co-chair of O'Reilly's Open Source Convention and leads the development of conference software at O'Reilly. He has also been XML.com managing editor, a Debian developer, and a GNOME contributor.

www.oreilly.com

US $34.99 **CAN $34.99**

ISBN: 978-0-596-51877-6

Safari Books Online

Free online edition for 45 days with purchase of this book. Details on last page.

Learning Rails

Other resources from O'Reilly

Related titles

Enterprise Rails
Ruby Cookbook™
Ajax on Rails
The Ruby Programming Language

oreilly.com

oreilly.com is more than a complete catalog of O'Reilly books. You'll also find links to news, events, articles, weblogs, sample chapters, and code examples.

oreillynet.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences

O'Reilly Media brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.

Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today for free.

Learning Rails

Simon St.Laurent and Edd Dumbill

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Learning Rails
by Simon St.Laurent and Edd Dumbill

Copyright © 2009 Simon St.Laurent and Edd Dumbill. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safari.oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides
Production Editor: Sarah Schneider
Production Services: Appingo, Inc.
Indexer: Seth Maislin
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Jessamyn Read

Printing History:
November 2008: First Edition.

O'Reilly and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Learning Rails*, the image of tarpans, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-51877-6

[M]

1226091111

Table of Contents

Preface xiii

1. Starting Up Ruby on Rails 1
Getting Started in the Online Cloud: Heroku 1
Getting Started with Instant Rails 5
Getting Started at the Command Line 8
Starting Up Rails 10
Dodging Database Issues 12
What Server Is That? 13
Test Your Knowledge 14
Quiz 14
Answers 14

2. Rails on the Web 15
Creating Your Own View 15
What Are All Those Folders? 19
Adding Some Data 20
How Hello World Works 22
Protecting Your View from the Controller 24
Parentheses Are (Usually) Optional 25
Adding Logic to the View 26
Test Your Knowledge 28
Quiz 28
Answers 28

3. Adding Web Style 29
I Want My CSS! 29
Layouts 32
Splitting View from Layout 32
Creating a Default Layout 34
Choosing a Layout from a Controller 35
Sharing Template Data with the Layout 37

Setting a Default Page 38
Test Your Knowledge 40
Quiz 40
Answers 40

4. Controlling Data Flow: Controllers and Models 41
Getting Started, Greeting Guests 41
Application Flow 46
Keeping Track: A Simple Guestbook 47
Connecting to a Database Through a Model 47
Connecting the Controller to the Model 50
Finding Data with ActiveRecord 54
Test Your Knowledge 56
Quiz 56
Answers 57

5. Accelerating Development with Scaffolding and REST 59
A First Look at Scaffolding 59
REST and Controller Best Practices 63
Websites and Web Applications 63
Toward a Cleaner Approach 65
Examining a RESTful Controller 66
Index: An Overview of Data 71
Show: Just One Row of Data 72
New: A Blank Set of Data Fields 73
Edit: Hand Me That Data, Please 74
Create: Save Something New 74
Put This Updated Record In 75
Destroy It 76
Escaping the REST Prison 77
Test Your Knowledge 78
Quiz 78
Answers 78

6. Presenting Models with Forms 81
More Than a Name on a Form 81
Generating HTML Forms with Scaffolding 82
Form As a Wrapper 87
Creating Text Fields and Text Areas 89
Creating Checkboxes 90
Creating Radio Buttons 92
Creating Selection Lists 94
Dates and Times 97

Labels 98
Creating Helper Methods 99
Letting Helper Methods Make Choices 101
A More Elegant Helper Method 102
Putting the Form Body in a Partial 102
Test Your Knowledge 104
Quiz 104
Answers 105

7. Strengthening Models with Validation 107
Without Validation 107
The Original Model 110
The Power of Declarative Validation 111
Managing Secrets 113
Customizing the Message 113
Limiting Choices 115
Testing Format with Regular Expressions 116
Seen It All Before 116
Numbers Only 117
A Place on the Calendar 118
Testing for Presence 119
Beyond Simple Declarations 119
Test It Only If 119
Do It Yourself 120
Test Your Knowledge 121
Quiz 121
Answers 121

8. Improving Forms 123
Adding a Picture by Uploading a File 123
File Upload Forms 124
Model and Migration Changes 125
Results 130
Standardizing Your Look with Form Builders 133
Supporting Your Own Field Types 133
Adding Automation 135
Integrating Form Builders and Styles 137
Test Your Knowledge 141
Quiz 141
Answers 142

9. Developing Model Relationships 143
Connecting Awards to Students 144

Establishing the Relationship 144
Supporting the Relationship 145
Guaranteeing a Relationship 148
Connecting Students to Awards 150
Removing Awards When Students Disappear 150
Counting Awards for Students 150
Nesting Awards in Students 153
Changing the Routing 153
Changing the Controller 154
Changing the Award Views 156
Connecting the Student Views 160
Is Nesting Worth It? 161
Many-to-Many: Connecting Students to Courses 162
Creating Tables 162
Connecting the Models 164
Adding to the Controllers 165
Adding Routing 167
Supporting the Relationship Through Views 167
What's Missing? 174
Test Your Knowledge 175
Quiz 175
Answers 175

10. Managing Databases with Migrations 177
What Migrations Offer You 177
Migration Basics 178
Migration Files 179
Running Migrations Forward and Backward 180
Inside Migrations 181
Working with Tables 182
Data Types 183
Working with Columns 184
Indexes 185
Other Opportunities 186
Test Your Knowledge 187
Quiz 187
Answers 187

11. Debugging 189
Creating Your Own Debugging Messages 189
Logging 190
Working with Rails from the Console 191
The Ruby Debugger 195

Test Your Knowledge 199
Quiz 199
Answers 199

12. **Testing** **201**
Test Mode 201
Setting Up a Test Database with Fixtures 202
Unit Testing 206
Functional Testing 212
Calling Controllers 214
Testing Responses 215
Dealing with Nested Resources 216
Integration Testing 217
Beyond the Basics 220
Test Your Knowledge 221
Quiz 221
Answers 221

13. **Sessions and Cookies** **223**
Getting Into and Out of Cookies 223
Storing Data Between Sessions 230
Test Your Knowledge 236
Quiz 236
Answers 236

14. **Users and Authentication** **237**
Installation 237
Storing User Data 238
Controlling Sessions 239
Classifying Users 246
More Options 252
Test Your Knowledge 252
Quiz 252
Answers 253

15. **Routing** **255**
Creating Routes to Interpret URIs 256
Specifying Routes with map.connect 256
A Domain Default with map.root 258
Route Order and Priority 259
Named Routes 259
Globbing 260
Regular Expressions and Routing 260

Mapping Resources 261
Nesting Resources 262
Checking the Map 263
Generating URIs from Views and Controllers 264
Pointing url_for in the Right Direction 264
Adding Options 265
Infinite Possibilities 266
Test Your Knowledge 267
Quiz 267
Answers 267

16. Creating Dynamic Interfaces with Rails and Ajax 269
Ajax Basics 269
Supporting Ajax with Rails 271
Rails as a Server API 271
Rails and the Client 272
Managing Enrollment through Ajax 273
Making the Form More Scriptable 273
Changing Courses without Changing Pages 277
Rethinking Logic 282
Moving Further into Ajax 283
Test Your Knowledge 283
Quiz 283
Answers 283

17. Mail in Rails 285
Sending Text Mail 285
Setup 285
Adjusting Routing for an email Method 286
Sending Email 287
Sending HTML Mail 291
Sending Complex HTML Email 294
Receiving Mail 299
Setup 299
Processing Messages 300
Test Your Knowledge 303
Quiz 303
Answers 303

18. Securing, Managing, and Deploying Your Rails Projects 305
Securing Your Application 305
SQL Injection 306
Cross-Site Scripting 306

Cross-Site Request Forgery (CSRF) 307
URL Hacking 308
Other Security Issues 308
Deploying Rails Applications 309
Changing to Production Mode 309
Database, Web, and App Servers 310
Walking Through a Passenger-Based Deployment 312
Deployment Directions 320
Test Your Knowledge 321
Quiz 321
Answers 321

19. Making the Most of Rails—And Beyond 323
Keep Up with Rails 323
Plug-ins 323
Ruby 324
Web Services 325
Explore Other Ruby Frameworks 325
Migrating Legacy Applications to Rails 326
Keep Exploring 327

A. An Incredibly Brief Introduction to Ruby 329

B. An Incredibly Brief Introduction to Relational Databases 349

C. An Incredibly Brief Guide to Regular Expressions 357

D. A Catalog of Helper Methods 367

E. Glossary 383

Index 399

Preface

Everyone cool seems to agree: Ruby on Rails is an amazing way to build web (or heck, Web 2.0) applications. Ruby is a powerful and flexible programming language, and Rails takes advantage of that flexibility to build a web application framework that takes care of a tremendous amount of work for the developer. Everything sounds great!

Except, well... all the Ruby on Rails books talk about this "Model-View-Controller" thing, and they start deep inside the application, close to the database, most of the time. From an experienced Rails developer's perspective, this makes sense—the framework's power lies largely in making it easy for developers to create a data model quickly, layer controller logic on top of that, and then, once all the hard work is done, put a thin layer of interface view on the very top. It's good programming style, and it makes for more robust applications. Advanced Ajax functionality seems to come almost for free!

From the point of view of someone learning Ruby on Rails, however, that race to show off Rails' power can be extremely painful. There's a lot of seemingly magical behavior in Rails that works wonderfully—until one of the incantations isn't quite right and figuring out what happened means unraveling all that work Rails did. Rails certainly makes it easier to work with databases and objects without spending forever thinking about them, but there are a lot of things to figure out before that ease becomes obvious.

If you'd rather learn Ruby on Rails more slowly, starting from pieces that are more familiar to the average web developer and then moving slowly into controllers and models, you're in the right place. You can start from the HTML you already likely know, and then move more deeply into Rails' many interlinked components.

Who This Book Is For

You've probably been working with the Web for long enough to know that writing web applications always seems more complicated than it should be. There are lots of parts to manage, along with lots of people to manage, and hopefully lots of visitors to please. Ruby on Rails has intrigued you as one possible solution to that situation.

You may be a designer who's moving toward application development or a developer who combines some design skills with some programming skills. You may be a

programmer who's familiar with HTML but who lacks the sense of grace needed to create beautiful design—that's a fair description of one of the authors of this book, anyway. Wherever you're from, whatever you do, you know the Web well and would like to learn how Rails can make your life easier.

The only mandatory technical prerequisite for reading this book is direct familiarity with HTML and a general sense of how programming works. You'll be inserting Ruby code into that HTML as a first step toward writing Ruby code directly, so understanding HTML is a key foundation. (If you don't know Ruby at all, you probably want to look over Appendix A or at least keep it handy for reference.)

Cascading Style Sheets (CSS) will help you make that HTML look a lot nicer, but it's not necessary for this book. Similarly, a sense of how JavaScript works may help. Experience with other templating languages (like PHP, ASP, and ASP.NET) can also help, but it isn't required.

You also need to be willing to work from the command line sometimes. The commands aren't terribly complicated, but they aren't (yet) completely hidden behind a graphical interface. Even Heroku, an online integrated development environment (IDE) for Rails, still has some necessary command-line features.

Who This Book Is Not For

We don't really want to cut anyone out of the possibility of reading this book, but there are a lot of people who aren't likely to enjoy it. Model-View-Controller purists will probably grind their teeth through the first few chapters, and people who insist that data structures are at the heart of a good application are going to have to wait an even longer time to see their hopes realized. If you consider HTML just a nuisance that programmers have to put up with, odds are good that this book isn't for you. Most of the other Ruby on Rails books, though, are written for people who want to start from the model!

Also, people who are convinced that Ruby and Rails are the one true way may have some problems with this book, which spends a fair amount of time warning readers about potential problems and confusions they need to avoid. Yes, once you've worked with Ruby and Rails for a while, their elegance is obvious. However, reaching that level of comfort and familiarity is often a difficult road. This book attempts to ease as many of those challenges as possible by describing them clearly.

What You'll Learn

Building a Ruby on Rails application requires mastering a complicated set of skills. You may find that—depending on how you're working with it, and who you're working with—you only need part of this tour. That's fine. Just go as far as you think you'll need.

At the beginning, you'll need to install Ruby on Rails. We'll explore different ways of doing this, with an emphasis on easier approaches to getting Ruby and Rails operational.

Next, we'll create a very simple Ruby on Rails application, with only a basic view and then a controller that does a very few things. From this foundation we'll explore ways to create a more sophisticated layout using a variety of tools, learning more about Ruby along the way.

Once we've learned how to present information, we'll take a closer look at controllers and what they can do. Forms processing is critical to most web applications, so we'll build a few forms and process their results, moving from the simple to the complex.

Forms can do interesting things without storing data, but after a while it's a lot more fun to have data that lasts for more than just a few moments. The next step is setting up a database to store information and figuring out how the magic of Rails' ActiveRecord makes it easy to create code that maps directly to database structures—without having to think too hard about database structures or SQL.

Once we have ActiveRecord up and running, we'll explore scaffolding and its possibilities. Rails scaffolding not only helps you build applications quickly, it helps you learn to build them well. The RESTful approach that Rails 2.0 chose to emphasize will make it simpler for you to create applications that are both attractive and maintainable. For purposes of illustration, using scaffolding also makes it easier to demonstrate one task at a time, which we hope will make it easier for you to understand what's happening.

Ideally, at this point you'll feel comfortable with slightly more complicated data models, and we'll take a look at applications that need to combine data in multiple tables. Mixing and matching data is at the heart of most web applications.

We'll also take a look at testing and debugging Rails code, a key factor in the framework's success. Migrations, which make it easy to modify your underlying data structures (and even roll back those changes if necessary), are another key part of Rails' approach to application maintainability.

The next step will be to add some common web applications elements like sessions and cookies, as well as authentication. Rails (sometimes with the help of plug-ins) can manage a lot of this work for you.

We'll also let Rails stretch its legs a bit, building more exciting Ajax applications and sending email messages. Finally, we'll show you one approach to bringing your Rails application to a wider public, deploying it with MySQL and Phusion Passenger, as well as exploring some other possibilities.

By the end of this tour, you should be comfortable with working in Ruby on Rails. You may not be a Rails guru yet, but you'll be ready to take advantage of all of the other resources out there for becoming one.

Ruby and Rails Style

It's definitely possible to write Ruby on Rails code in ways that look familiar to programmers from other languages. However, that code often isn't really idiomatic Ruby, as Ruby programmers have chosen other paths. In general, this book will always try to introduce new concepts using syntax that's likely to be familiar to developers from other environments, and then explain what the local idiom does. You'll learn to write idiomatic Ruby that way (if you want to), and at the same time you'll figure out how to read code from the Ruby pros.

We've tried to make sure that the code we present is understandable to those without a strong background in Ruby. Ruby itself is worth an introductory book (or several), but the Ruby code in a lot of Rails applications is simple, thanks to the hard work the framework's creators have already put into it. You may want to install Rails in Chapter 1, and then explore Appendix A, "A Quick Guide to Ruby," if you want some background before diving in.

Other Options

There are lots of different ways to learn Rails. Some people want to learn Ruby in detail before jumping into a framework that uses it. That's a perfectly good option, and if you want to start that way, you should explore:

- *Learning Ruby* (O'Reilly, 2007)
- *The Ruby Programming Language* (O'Reilly, 2008)
- *Ruby Pocket Reference* (O'Reilly, 2007)
- *Programming Ruby*, Third Edition (Pragmatic Programmers, 2008)

You may also want to supplement (or replace) this book with other books on Rails. If you want some other resources, you can explore:

- *Head First Rails* (O'Reilly, 2008), for a much more visual approach with exercises
- *Up and Running with Rails*, Second Edition (O'Reilly, 2008), for a very quick start
- *Simply Rails 2* (SitePoint, 2008) takes a similar approach to *Learning Rails*, but with different opinions and details
- *http://www.learningrails.com*, a site with free podcasts and screencasts for getting started in Rails
- *The Rails Way* (Addison-Wesley, 2007), a big-book reference approach for developers who already know their way
- *Rails Pocket Reference* (O'Reilly, 2008), a small-book reference
- *Agile Web Development with Rails*, Third Edition (Pragmatic Programmers, 2008), for a detailed explanation of a wide range of features.

- *Enterprise Rails* (O'Reilly, 2008), for building large-scale applications
- *Advanced Rails* (O'Reilly, 2008), for when you want to move to the next level

You'll want to make sure that whatever books or online documentation you use covers Rails 2.0 or later. Rails' perpetual evolution has unfortunately made it dangerous to use a lot of formerly great but now dated material. (Some of it works, some of it doesn't.)

Rails Versions

The Rails team is perpetually improving Rails and releasing new versions. This book was written using Rails 2.0 and 2.1, and all examples have been tested in 2.1. Rails 2.2 will be out soon, and it doesn't look like there are any major changes coming beyond a few noted in the text. We'll post updates on new versions at *http://www.excursionsonrails.com*.

If You Have Problems Making Examples Work

When you're starting to use a new framework, error messages can be hard, even impossible, to decipher. We've included occasional notes in the book about particular errors you might see, but it seems very normal for different people to encounter different errors as they work through examples. Sometimes it's the result of skipping a step or entering code just a little differently than it was in the book. It's probably not the result of a problem in Rails itself, even if the error message seems to come from deep in the framework. That isn't likely an error in the framework, but much more likely a problem the framework is having in figuring out how to deal with the unexpected code it just encountered.

If you find yourself stuck, here are a few things you should check:

What version of Ruby are you running?
: You can check by entering `ruby -v`. All of the examples in this book were written with Ruby 1.8.6. Older versions of Ruby may cause problems for Rails, and the 1.9 versions add features, but may create new issues as well. Chapter 1 explores how to install Ruby, but you may need to find documentation specific to your specific operating system and environment.

What version of Rails are you running?
: You can check by running `rails -v`. While you should be able to use the examples here with any version of Rails 2.*x*, the examples, including the ones you can download from the book's site, were built on Rails 2.1.0. If you're running a different version, especially an earlier version, you may encounter problems. (While a few of the examples here may run on versions of Rails older than 2.0, most of them will encounter major problems quickly.)

Are you calling the program the right way?

Linux and Mac OS X both use a forward slash, `/`, as a directory separator, whereas Windows uses a backslash, `\`. This book uses the forward slash, but if you're in Windows, you may need to use the backslash.

Is the database connected?

By default, Rails expects you to have SQLite up and running, though some installations use MySQL or other databases. If you're getting errors that have "sql" in them somewhere, it's probably the database. For simple applications that aren't calling a database, check the instructions at the end of Chapter 1 for telling Rails not to look for a database. For more complex applications where your application expects a database, check that the database is installed and running, that the settings in *database.yml* are correct, and that the permissions, if any, are set correctly.

Are all of the pieces there?

Most of the time, assembling a Rails application, even a simple one, requires modifying multiple files—at least a view and a controller. If you've only built a controller, you're missing a key piece you need to see your results; if you've only built a view, you need a controller to call it. As you build more and more complex applications, you'll need to make sure you've considered routing, models, and maybe even configuration and plug-ins. What looks like a simple call in one part of the application may depend on pieces elsewhere.

Eventually, you'll know what kinds of problems specific missing pieces cause, but at least at first, try to make sure you've entered complete examples before running them.

It's also possible to have files present but with the wrong permissions set. If you know a file is there, but Rails can't seem to get to it, check to make sure that permissions are set correctly.

Is everything named correctly?

Rails depends on naming conventions to establish connections between data and code without you having to specify them explicitly. This works wonderfully, until you have a typo somewhere obscure. Rails also relies on a number of Ruby conventions for variables, prefacing instance variables with `@` or symbols with `:`. These special characters make a big difference, so make sure they're correct.

Is the Ruby syntax right?

If you get syntax errors, or sometimes even if you get a `nil` object error, you may have an extra space, missing bracket, or similar issue. Ruby syntax is extremely flexible, so you can usually ignore the discipline of brackets, parentheses, or spaces—but sometimes it really does matter.

Did the authors just plain screw up?

Obviously, we're working hard to ensure that all of the code in this book runs smoothly the first time, but it's possible that an error crept through. You'll want to check the errata, described in the next section, and download sample code, which will be updated for errata.

It's tempting to try Googling errors to find a quick fix. Unfortunately, the issues just described are more likely to be the problem than something else that has clear documentation. The Rails API documentation might be helpful at times, especially if you're experimenting with extending an example. There shouldn't be much out there, though, beyond the book example files themselves that you can download to fix an example.

If You Like (or Don't Like) This Book

If you like—or don't like—this book, by all means, please let people know. Amazon reviews are one popular way to share your happiness (or lack of happiness), or you can leave reviews on the site for this book:

http://www.oreilly.com/catalog/9780596518776/

There's also a link to errata there. Errata gives readers a way to let us know about typos, errors, and other problems with the book. The errata will be visible on the page immediately, and we'll confirm it after checking it out. O'Reilly can also fix errata in future printings of the book and on Safari, making for a better reader experience pretty quickly.

We hope to keep this book updated for future versions of Rails and will also incorporate suggestions and complaints into future editions.

Conventions Used in This Book

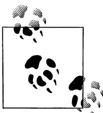
The following font conventions are used in this book:

Italic
: Indicates pathnames, filenames, and program names; Internet addresses, such as domain names and URLs; and new items where they are defined.

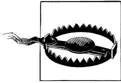
`Constant width`
: Indicates command lines and options that should be typed verbatim; names and keywords in programs, including method names, variable names, and class names; and HTML element tags.

`Constant width bold`
: Indicates emphasis in program code lines.

`Constant width italic`
: Indicates text that should be replaced with user-supplied values.



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

The code examples for this book, which are available from *http://oreilly.com/catalog/9780596518776/*, come in two forms. One is a set of examples, organized by chapter, with each example numbered and named. These examples are referenced from the relevant chapter. The other form is a dump of all the code from the book, in the order it was presented in the book. That can be helpful if you need a line that didn't make it into the final example, or if you want to cut and paste pieces as you walk through the examples. Hopefully, the code will help you learn.

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Learning Rails* by Simon St.Laurent and Edd Dumbill. Copyright 2009 Simon St.Laurent and Edd Dumbill, 978-0-596-51877-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made a few mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the U.S. or Canada)
707-829-0515 (international/local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://oreilly.com/catalog/9780596518776/

There is also a supporting page for the book, including screencasts, installation help, and more, at:

http://excursionsonrails.com/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com

Acknowledgments

Thanks to Mike Loukides for thinking that Rails could use a new and different approach, and for supporting this project along the way. Tech reviewers Gregg Pollack, Shelley Powers, Mike Fitzgerald, Eric Berry, David Schruth, Mike Hendrickson, and Mark Levitt all helped improve the book tremendously. The rubyonrails-talk group provided regular inspiration, as did the screencasts and podcasts at *http://railscasts.com* and *http://railsenvy.com*.

Edd Dumbill wishes to thank his lovely children, Thomas, Katherine and Peter, for bashing earnestly on the keyboard, and his coauthor, Simon St.Laurent, for his patient encouragement in writing this book.

Simon St.Laurent wants to thank Angelika St.Laurent for her support over the course of writing this, even when it interfered with dinner, and Sungiva St.Laurent for her loudly shouted suggestions. Simon would also like to thank Edd Dumbill for his initial encouragement and for making this book possible.

We'd like to thank Sarah Schneider, for seeing this book through production, as well as Mark Jewett and Virginia Ogozalek at Appingo for their work on it. Jessamyn Read made the figures much more appealing, and Seth Maislin created the index.

CHAPTER 1
Starting Up Ruby on Rails

Before you can use Rails, you have to install it. Even if it's already installed on your computer, or you opt to use a web-based development environment, there are a few things you'll need to do to make it actually do something visible. In this chapter, we'll take a look at some ways of installing Ruby, Rails, and the supporting infrastructure, and get a first, rather trivial project up and running.

To get you started, we'll set up three different environments for running Rails: Heroku (the fastest option for getting started, which lets you develop applications online), Instant Rails on Windows, and the classic command-line version. Feel very welcome to jump to whatever pieces of this section interest you and skip past those that don't. Once the software is working, we'll generate the basic Rails application, which will at least let you know if Rails is working.

If you want to jump into learning Rails without getting hung up on installation, Heroku is likely your easiest approach, with Instant Rails a close second for Windows users. They both create an insulated environment separate from the rest of what your computer might have, and they require minimal configuration. If the classic command-line approach doesn't appeal to you, or causes you problems, definitely give Heroku or Instant Rails a try. However you decide to set up Rails, in the end you're going to need to install a structure like that shown in Figure 1-1.

All of these options are free. You don't need to spend any money to use Rails, unless maybe you feel like buying a nice text editor.

Getting Started in the Online Cloud: Heroku

Rails is hot, and cloud computing is hot. Why maintain your own server when you could have Amazon run your applications in a "cloud" of servers? And why not develop your Rails applications on the Web, instead of dealing with configuration details on your own computer?

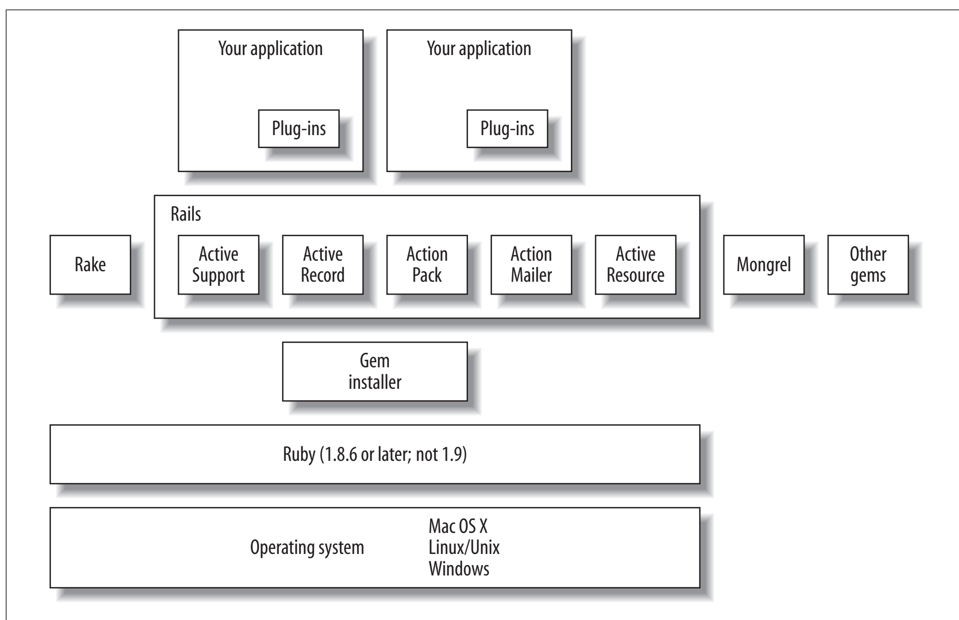


Figure 1-1. The many components of a Rails installation

The folks at Heroku (*http://heroku.com*) have created a web-based development framework for Ruby on Rails applications. Right now, it's free, though it's a beta, and betas are always risky. Its business model seems to lead to the paid hosting of running apps, so the development part might even stay free if all goes well.

Rather than downloading Heroku, you sign up. It's a little buried in the beta version, but if you visit *http://heroku.com* and click on Login, you'll find a "Sign up" link. It's a limited beta, but the wait times have been pretty reasonable. (As of this writing, it says "less than a day.") Once you're signed in, you can go to "My apps" and click on "Create new app." A few sections later, you'll see your application development environment, something like Figure 1-2.

Heroku's Code tab, where you start, looks pretty much like a set of file directories. You can navigate directories and open and edit files using Heroku's built-in editor. The downward-pointing arrows next to the directory names bring up menus for creating and managing files. Below the directory list, the Revisions >> link lets you see what you've changed lately and commit changes to keep more precise track of what's changed.

Across the top, the Data tab gives you access to the databases you'll be creating, and Logs takes you to the log files recording the activities of your application. The fast-forward button (>>) on the top right will actually run your application. By default, it'll look like a customized version of Figure 1-10, made a little more specific to Heroku, as shown in Figure 1-3.

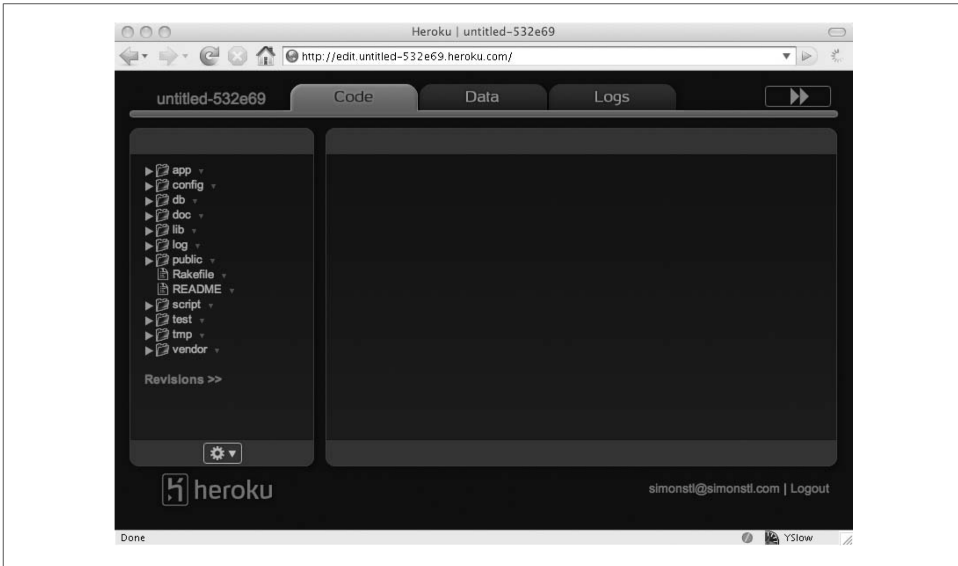


Figure 1-2. A brand new Rails application development environment, in Heroku

At the bottom of Figure 1-3, you can see the rewind button (<<), which will take you back to the development environment. There's one more piece you need to find before proceeding into Rails development, which is the gear button near the bottom of the left column. If you click on it, as you can see in Figure 1-4, you'll get a menu of options for performing tasks you would normally have done from the command line.

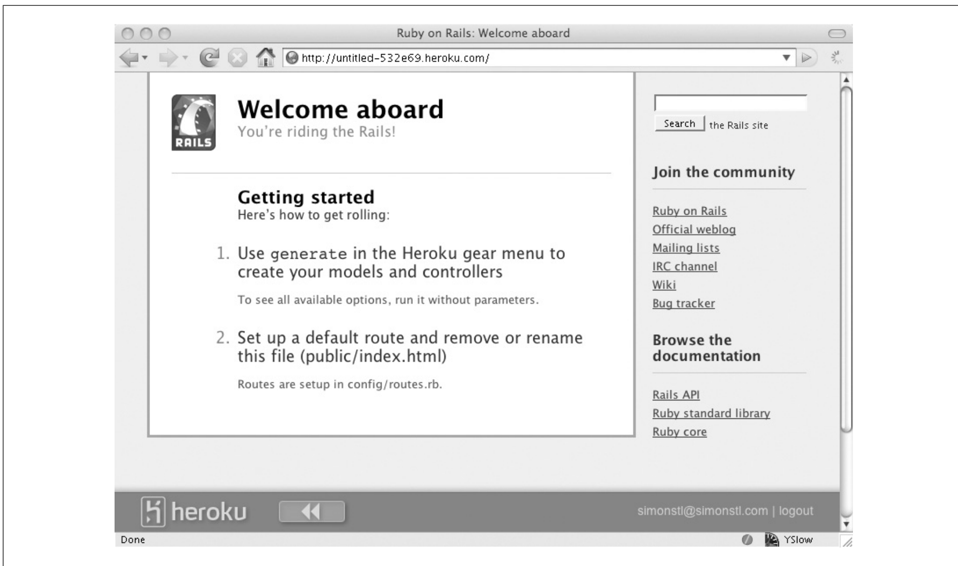


Figure 1-3. Signs that Rails is running

Figure 1-4. The Heroku gear menu

The gear menu gives you access to many of the Rails tools that are normally used from the command line. Most of the rest of this book describes using them that way, because both Instant Rails and more traditional installs usually work that way. Just remember that in Heroku, `ruby script/server` means clicking the >> button, and that `script/generate`, `script/console`, and `rake` are run as the Generate, Console, and Rake commands from the gear menu. Gems and plug-ins, which you'll use to extend Rails, can be installed from the `Gems and Plugins>>` menu item under the *vendor* directory.

And don't worry—you're not locked into Heroku. You can develop applications elsewhere and bring them in, or develop them in Heroku and export them. The settings for your application, shown in Figure 1-5, also let you make your apps public or private, make snapshots, or destroy your application if you don't like it.

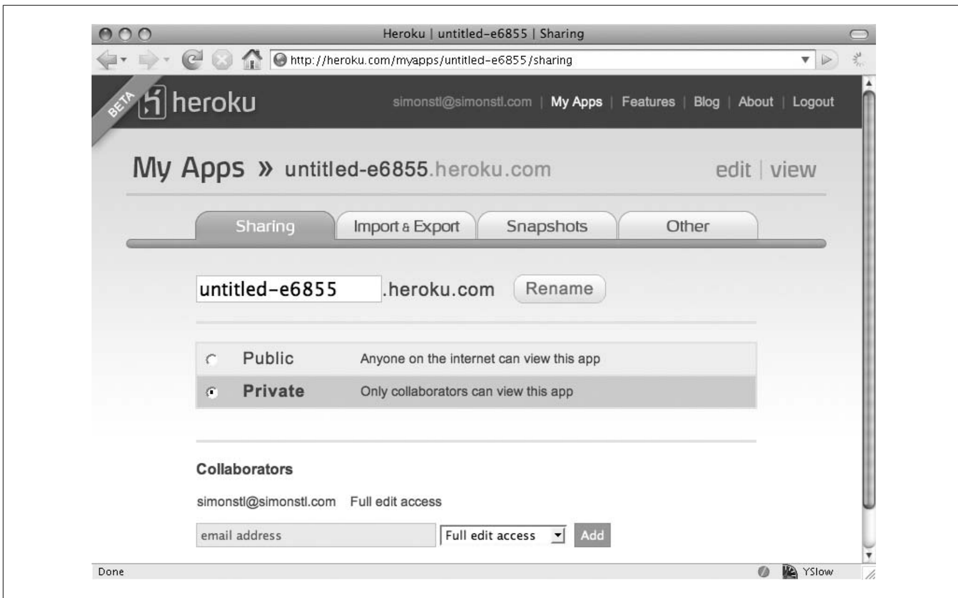


Figure 1-5. Application settings in Heroku

If you're looking to experiment with Rails and want to get started quickly, Heroku can be a great option, whether or not you want your application to reside "in the cloud."

If you like Heroku but find the web interface constraining, a command-line set of tools and an API are also available. You can find more information at *http://rubyforge.org/projects/heroku/* or at *http://technicalpickles.com/posts/playing-with-heroku*.

Getting Started with Instant Rails

Instant Rails is a single-install environment for Windows that includes the parts you need to get started building Rails applications. Installing it is trivial. Get the latest version from *http://instantrails.rubyforge.org/wiki/wiki.pl* and unzip the file wherever you'd like it to go. There is no real install program.

Once you've unzipped Instant Rails, you can start to develop your Rails application by opening *InstantRails.exe*. You'll see the basic screen shown in Figure 1-6.

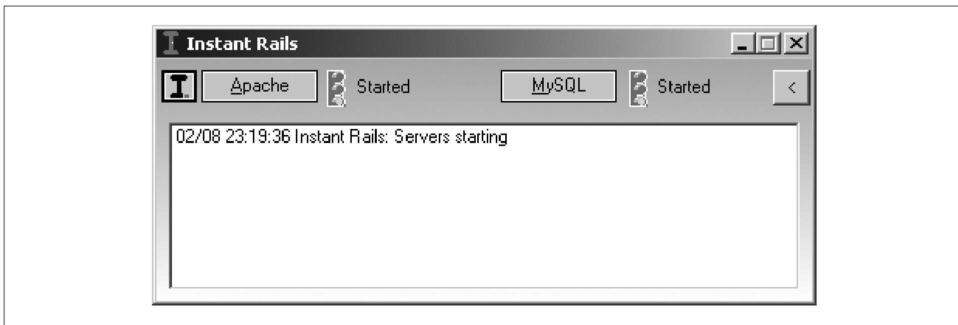
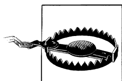


Figure 1-6. The main Instant Rails screen



If you're running Windows Vista, you may get a warning that "The publisher could not be verified." If you want to run Instant Rails, you'll need to click the Run button. (To turn off the warning, uncheck the "Always ask before opening this file" box.)

You may also get a warning about Apache or Ruby also being blocked from accepting incoming network requests. You'll have to unblock those, too.

To create a new application, click on the "I" button to the left of the Apache button, and choose Rails Applications and then Manage Rails Applications.... You'll see Figure 1-7, the dialog for managing Rails applications, appear.

To create a new Rails application (beyond the samples that come with Instant Rails), click on the Create New Rails App... button at the lower left of Figure 1-7. You might expect to get a dialog box, but surprise! You'll get something like Figure 1-8.

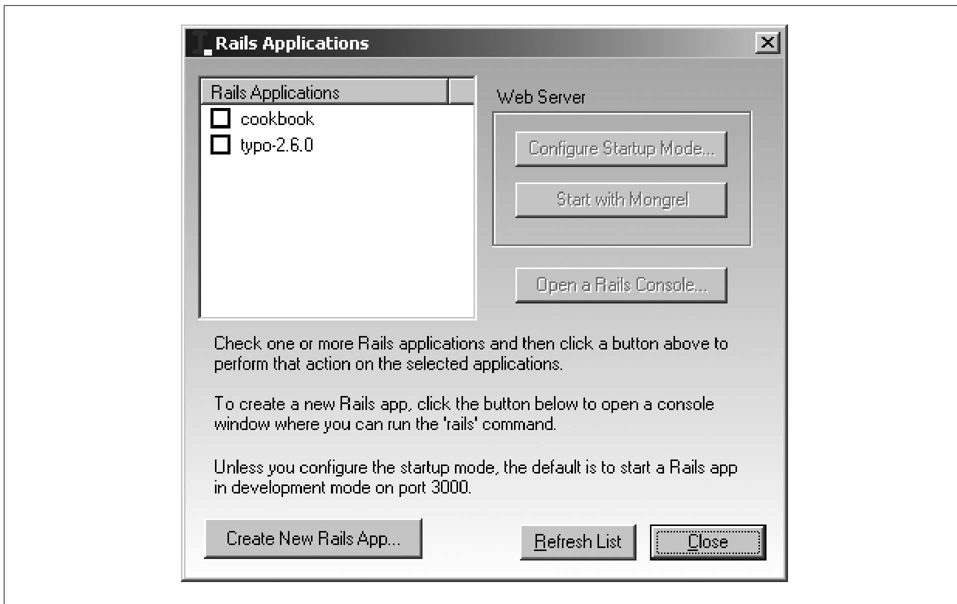


Figure 1-7. The Rails Applications dialog, where you can start and stop applications, as well as create new ones

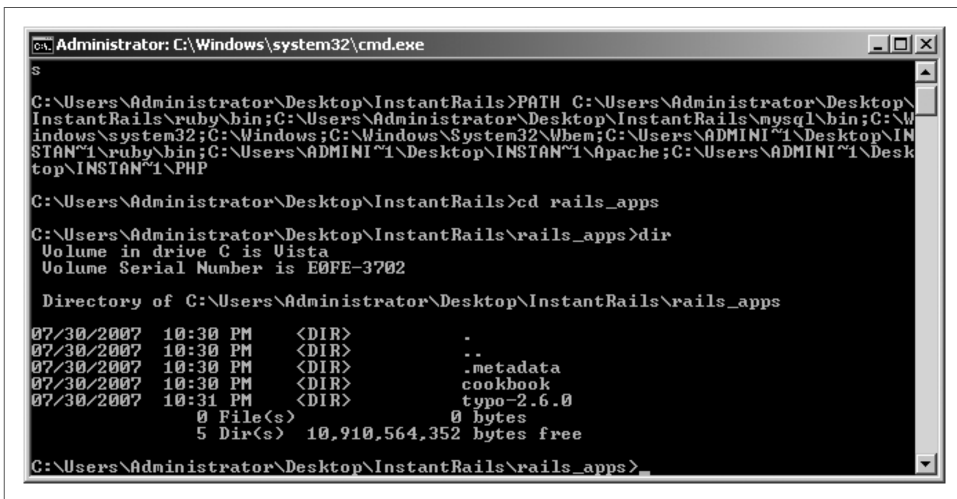


Figure 1-8. A configured command-line environment

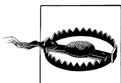
To create a new Rails application here, just type `rails hello` at the command line. As you'll see in the next section, Rails will let you know it's created a lot of files.

To actually start the application, you'll need to go back to the Rails Applications dialog, click Refresh List, and then check your *hello* application as shown in Figure 1-9.

Figure 1-9. Selecting your new application before starting it

If you click on Start with Mongrel, another command-line window will open, and the Mongrel web server will start running on port 3000. To see what it's running, open a web browser and visit *http://localhost:3000/*. You'll see the advice given in Figure 1-10.

From here, you can proceed with customizing the *hello* project so that it actually does something. Most of your work will be in the text editor or IDE of your choice, along with the command line. To get to the command line to work more with Rails, you'll want to click the "I" on the main Instant Rails dialog box again, and choose Rails Applications and Open Ruby Console Window. To get to the files in your applications, choose Rails Applications and Open Windows Explorer.



You can't just open a normal command-prompt window, since Instant Rails' all-in-one approach leaves the normal "DOS box" knowing nothing about Rails. Any time that you want to connect with your Ruby application from the command line, you *must* go through Instant Rails' Open Ruby Console Window button.

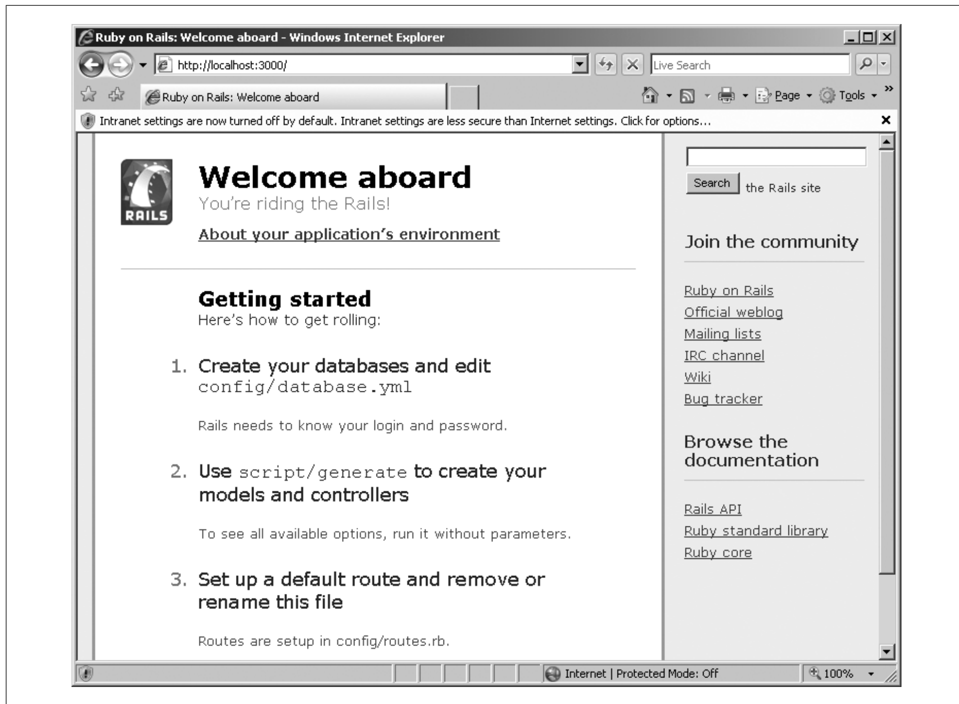
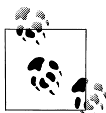


Figure 1-10. The Rails welcome page

You can upgrade gem-based components within Instant Rails using `gem install rails` at its command line. For more detailed information on upgrading the version of Rails inside Instant Rails to the current version, visit *http://excursionsonrails.com/*.

Getting Started at the Command Line

Instant Rails and Heroku may offer relatively easy ways to create and manage a Rails application, but they're definitely not necessary. Installing Rails by hand requires installing Ruby, installing Gems, and then installing Rails. You will eventually also need to install SQLite, MySQL, or another relational database, though SQLite is already present on the Mac and in many Linux distributions.

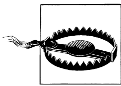


If you're wondering how to find this "command line," you need to find a terminal application. On the Mac, it's called Terminal, and it's in the *Utilities* folder of *Applications*. Linux terminals vary, but it's probably gnome-terminal or kterm. On Windows, it's the Command Prompt, *cmd.exe*. If you've never used a command line, you may want to get a quick reference guide for your operating system that covers it.

Ruby comes standard on a number of Linux and Macintosh platforms. To see whether it's there, and what version it has, enter `ruby -v` at the command prompt. You'll want Ruby 1.8.6 or later, so you may need to update it to a more recent version:

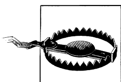
- On Mac OS X, Leopard (10.5) includes Ruby 1.8.6, but the previous version of OS X included Ruby 1.8.2. If you're on Tiger (10.4) or an earlier version of OS X, you'll need to update Ruby itself, a challenge that's beyond the scope of this book. You may want to investigate MacPorts, and the directions at *http://nowiknow.wordpress.com/2007/10/07/install-ruby-on-rails-for-mac/*. For a more comprehensive installation, explore *http://paulsturgess.co.uk/articles/show/46*.
- For Windows, the One-Click Ruby Installer (*http://rubyinstaller.rubyforge.org/wiki/wiki.pl*) is probably your easiest option, though there are other alternatives, including Cygwin (*http://www.cygwin.com/*), which brings a lot of the Unix environment to Windows.
- Most distributions of Linux include Ruby, but you'll want to use your package manager to make sure it's updated to 1.8.6. Some, notably Ubuntu and Debian, will name the `gem` command `gem1.8`.

For more on how to install Ruby on a variety of platforms, see *http://www.ruby-lang.org/en/downloads/*.



You don't need to update Ruby to version 1.9—indeed, it's better if you don't, at this point.

Gems is also starting to come standard on a number platforms, most recently on Mac OS X Leopard, but if you need to install Gems, see the RubyGems User Guide's instructions at *http://www.rubygems.org/read/chapter/3*.



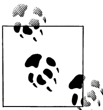
If you use MacPorts, apt-get, or a similar package installer, you may want to use it only to install Ruby, and then proceed from the command line. You certainly can install Gems and Rails with these tools, but Gems can update itself, which can make for very confusing package update issues.

Once you have Gems installed, Rails is just a command away:

```
~ simonstl$ sudo gem install rails
Password:
Successfully installed rake-0.8.1
Successfully installed activesupport-2.1.0
Successfully installed activerecord-2.1.0
Successfully installed actionpack-2.1.0
Successfully installed actionmailer-2.1.0
Successfully installed activeresource-2.1.0
Successfully installed rails-2.1.0
7 gems installed
```

```
Installing ri documentation for rake-0.8.1...
Installing ri documentation for activesupport-2.1.0...
Installing ri documentation for activerecord-2.1.0...
Installing ri documentation for actionpack-2.1.0...
Installing ri documentation for actionmailer-2.1.0...
Installing ri documentation for activeresource-2.1.0...
Installing RDoc documentation for rake-0.8.1...
Installing RDoc documentation for activesupport-2.1.0...
Installing RDoc documentation for activerecord-2.1.0...
Installing RDoc documentation for actionpack-2.1.0...
Installing RDoc documentation for actionmailer-2.1.0...
Installing RDoc documentation for activeresource-2.1.0...
```

You only need to use `sudo`, which gives your command the power of the root (administrative) account, if you're working in an environment that requires root access for the installation—otherwise, you can just type `gem install rails`. That will install the latest version of Rails, which may be more recent than 2.1, as well as all of its dependencies. (To see which version of Rails is installed, enter `rails -v` at the command line.)



Mac OS X Leopard (10.5) comes with Rails 1.2.3 installed. You'll definitely need to update Rails to version 2.1, as shown earlier, to work with the rest of this book. You'll also probably need to keep an eye on future updates from Apple that could change Rails on you, and maybe even lock down Rails versions in your critical applications with the `rake` tool's `freeze` task.

If you're ever wondering which gems (and which versions of gems) are installed, type `gem list --local`. For more information on gems, just type `gem`, or visit *http://rubygems.rubyforge.org*.

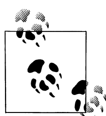
There are a few gems you may want to install, though these come preinstalled on Mac OS X 10.5. To install the Mongrel app server, run `sudo gem install mongrel`. To install the Ruby bindings for SQLite, run `sudo gem install sqlite3-ruby`. (You'll still need to install SQLite 3.)

You can see the documentation that gems have installed by running the command `gem server`, and visiting the URL (usually *http://localhost:8808*) that command reports. When you're done, you can turn off the server with Ctrl-C.

Starting Up Rails

Once you have Rails installed, you can create a Rails application easily from the command line:

```
~ $ rails hello
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
...
      create  public/images/rails.png
      create  public/javascripts/prototype.js
      create  public/javascripts/effects.js
      create  public/javascripts/dragdrop.js
      create  public/javascripts/controls.js
      create  public/javascripts/application.js
      create  doc/README_FOR_APP
      create  log/server.log
      create  log/production.log
      create  log/development.log
      create  log/test.log
```

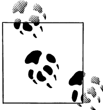


Rails application directories are just ordinary directories. You can move them, obliterate them and start over, or do whatever you need to do with ordinary file-management tools. Each application directory is also completely independent—the general "Rails environment" just generates these applications.

To start Rails, you'll need to move into the directory you just created—`cd hello`—and then issue your first command to get the Mongrel server busy running your application:

```
~ $ ruby script/server
=> Booting Mongrel (use 'script/server webrick' to force WEBrick)
=> Rails application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
** Starting Mongrel listening at 0.0.0.0:3000
** Starting Rails with development environment...
** Rails loaded.
** Loading any Rails specific GemPlugins
** Signals ready.  TERM => stop.  USR2 => restart.  INT => stop (no restart).
** Rails signals registered.  HUP => reload (without restart).  It might not work
well.
** Mongrel available at 0.0.0.0:3000
** Use CTRL-C to stop.
```

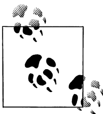
Rails is now running, and you can watch any errors it encounters through the extensive logging you'll see in this window.



On most Linux and Mac systems, you can leave off the `ruby` part—`script/server` will do. And you should note that by default, `script/server` binds only to `localhost`, and the application isn't visible from other computers. Normally, that's a security feature, not a bug, though you can specify an address for the server to use with the `-b` option (and `-p` for a specific port) if you want to make it visible.

For more details on options for using `script/server`, just enter `ruby script/server -h`.

If you now visit *http://localhost:3000*, you'll see the same welcome screen shown previously in Figures 1-3 and 1-10. When you're ready to stop Rails, you can just press Ctrl-C.



You really only need to stop Rails when you're done developing, if then. In development mode, you can make all the changes you want to your application with the server running, and you won't have to restart the server to see them.

Dodging Database Issues

By default, Rails 2.0 and later expects every application to have a database behind it. (That's why Figures 1-3 and 1-10 refer to configuring databases at the start.) That expectation makes it a little difficult to get started with Rails, so it can be a good idea to either make sure that SQLite is installed or turn off the features that will call a database, at least at first.

Rails 2.0.2 and later versions use SQLite as the default database, and connects to it much more automatically. If you're running an operating system that includes SQLite—such as many versions of Linux and Mac OS X 10.4 or later—you can skip this section. (You can also skip it if you installed Instant Rails or are using Heroku, where the databases are already running.) To check that it's available, you can run `sqlite3 -help`. If that returns a friendly help message, you're set. You can just run `rake db:create` or `rake db:migrate` from the command line before running your application, and that will perform the necessary database setup. If the help message doesn't come up, installing SQLite would be a good idea. (For more on SQLite, see *http://www.sqlite.org/*.)

If you decide to postpone database installation and get weird errors that look like your application can't find a database, and you weren't expecting it to need one, then you should turn off the database connection. The key to doing this is the *environment.rb* file, which you'll find in the *config* directory. About halfway down the file, you'll find:

```
# Skip frameworks you're not going to use (only works if using vendor/rails).
# To use Rails without a database, you must remove the ActiveRecord framework
# config.frameworks -= [ :active_record, :active_resource, :action_mailer ]
```

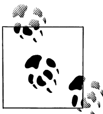
To turn off Rails' demand for a database, just remove the highlighted # symbol in front of `config.frameworks`. You need to do this *before* you start up Rails with `script/server`.

In development mode (which is where you start), you can change code on the fly and Rails will immediately reflect your changes, but this doesn't apply to configuration files. They only get loaded when Rails starts up. If you need to change any configuration files, stop your application and then start it again after you've saved the change.

We'll come back to Rails' powerful database-centric core after taking a closer look at how it interacts with the Web.

What Server Is That?

You might wonder what server is running your Rails application—after all, nothing so far has required any configuration, despite the fairly complicated usual installs needed to get web programming environments to run. You can continue without knowing (until it's time for real deployment), but if you're curious, here are the details.



Heroku doesn't say which server it's using, but fortunately that doesn't matter very much for getting things done.

You may be familiar with Apache or Microsoft's Internet Information Server (IIS), but neither of those web servers is probably running these Rails programs. (They can run Rails, but unless you took a very different path for installation, they're not running yet.) Instead, your programs are probably running in Mongrel. The Rails 2.0 command line uses Mongrel. (In earlier versions of Rails, running the application from the command line in the usual way started up an instance of WEBrick.) Instant Rails uses Apache with Mongrel actually interacting with Rails behind it. WEBrick and Mongrel come with slightly different priorities:

WEBrick (http://www.webrick.org/)
: WEBrick is written in Ruby and bundled with recent releases of Ruby. It's very convenient for Ruby development, with or without Rails. It's an excellent testing server, but not designed for large scale deployment.

Mongrel (http://mongrel.rubyforge.org/)
: Mongrel is a highly optimized server written in Ruby that "does the bare minimum necessary to serve a Ruby application." It's designed to be as absolutely fast as possible and is often used in conjunction with Apache on production web servers.

For development work, you'll likely run at least one of these servers on your local machine, probably on an odd port, like 3000, instead of the traditional default web server port of 80. For deployment, as described in Chapter 18, you'll probably use Apache (with Mongrel or Passenger behind it) or lighttpd. (You can deploy Rails on Windows with Mongrel or IIS, but it's rarely the most efficient approach.)

If you've never used Ruby before, now would be a good time to explore Appendix A, which teaches some key components of the language inside of a very simple Rails application.

Test Your Knowledge

Quiz

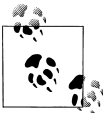
1. What is cloud computing and how can it help you develop in Rails?
2. What's the name of the Ruby application packaging utility and how do you install Rails with it?
3. In what instances would you avoid WEBrick?
4. Why should you install a particular version of Ruby on your platform when Ruby already comes installed?

Answers

1. Cloud computing provides access to virtual servers. You don't even have to know which server is hosting which part of the application. Mostly, you'll consider deploying your Rails applications on cloud computing environments, but Heroku offers you a chance to develop your applications through a web interface that runs on Amazon's EC2 cloud computing platform.
2. RubyGems, or just "gems," which is run with the `gem` command, is Ruby's application packager. To install the latest version of Rails and all its dependencies, just type `gem install rails`.
3. WEBrick is great for testing your Rails applications, but definitely not the best choice for deployments where performance matters.
4. Rails is still running on version 1.8 of Ruby, not the latest 1.9, but it requires features in the more recent versions of 1.8, notably 1.8.6.

CHAPTER 2
Rails on the Web

Now that you have Rails installed (or have signed into Heroku), it's time to make Rails do something—not necessarily very much yet, but enough to show you what happens when you make a call to a Rails application, and enough to let you do something to respond when those calls come in. There's a long tradition in computer books of starting out with a program that says "hello" to the programmer. We'll follow that tradition and pursue it a bit further to make clear how Rails can work with HTML. You're welcome, of course, to make Rails say whatever you'd like.



The work in this chapter depends on the *hello* application created in Chapter 1. If you didn't create one, go back and explore the directions given there. If you're working with Heroku, any new application will do. You can also find the files for the first demonstration in *ch02/hello001* of the downloadable code.

Creating Your Own View

Saying "hello" is a simple thing, focused exclusively on putting a message on a screen. To get started, we can post that message using a view including HTML that will get sent to the browser.

Rails actually won't let you create views directly. Its controller-centric perspective requires that views be associated with controllers. While that might seem like a bit of an imposition, it's not too hard to work around.

Creating anything in Rails requires going to the command line:

- In Instant Rails, you'll want to click the "I" on the main Instant Rails dialog box again, and choose Rails Applications and Open Ruby Console Window..., and then type `cd hello`.

- In Heroku, click on the gear menu and select Generate. (You can skip the `ruby script/generate` part below and just enter `controller Hello index` into the field Heroku presents.)
- If you're using another environment, open a terminal or command window and go to the home directory of your Rails application.

Then type:

```
ruby script/generate controller Hello index
```

The `script/generate` part of this command is calling a program, `generate`, in the *script* directory of the application. The first argument, `controller`, specifies that it should generate code for a controller, in this case named `Hello`, the second argument. Finally, including `index` at the end requests a view named index, bound to the hello controller.

On Linux and the Mac, you can generally leave off the `ruby` at the start of `script/generate` and similar commands.

Model-View-Controller

"You keep talking about views, controllers, and models. What is all that?"

It's a bit of programmer-speak: Model-View-Controller, or MVC, is an old idea that got its start in the Smalltalk programming world of the 1970s. The *model* is the underlying data structure, specific to the task the program is addressing; *controllers* manage the flow of data into and out of those objects; and *views* present the information provided by those controllers to users.

MVC is an excellent approach for building maintainable applications, as each layer keeps its logic to itself. Views might include a bit of code for presenting the data from the controller, but most of the logic for moving information around should be kept in the controller, and logic about data structures should be kept in the model. If you want to change how something looks, but not change the logic or the data structures, you can just create a new view, without disrupting everything underneath it.

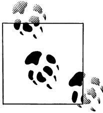
As you see more of Rails, in this book and elsewhere, you'll probably come to appreciate MVC's virtues, though it can seem confusing and constraining at first. Chapter 4 will explain how Rails uses MVC in more detail.

You'll see something like:

```
1       exists  app/controllers/
2       exists  app/helpers/
3       create  app/views/hello
4       exists  test/functional/
5       create  app/controllers/hello_controller.rb
```

```
6       create   test/functional/hello_controller_test.rb
7       create   app/helpers/hello_helper.rb
8       create   app/views/hello/index.html.erb
```

The lines starting with `exists` reflect directories or files that the generator could have created, but that were already there. The `create` entries identify directories and files that the generator created itself. You'll see a new *views* directory in line 3, a controller in line 5, a template for creating tests for that controller in line 6, a helper in line 7, and the index file (*index.html.erb*) we requested in line 8. (The *.rb* file extension is the conventional extension for Ruby files; *.erb* is the common extension for Embedded Ruby files.)



If you foul up a `script/generate` command, you can issue `script/destroy` to have Rails try to fix your mistakes.

That index file is now available to the application. Run `ruby script/server` (or click the play button in Heroku) to get it going, and then take a look at *hello* in the application. Figure 2-1 shows what Rails created to start with.

Figure 2-1. The generated index file identifies its home

This isn't pretty, but there's already something to learn here. Note that the URL that brought up this page is *http://localhost:3000/hello/*. As the page itself says, though, the file is in *app/views/hello/index.html.erb*. There's a web server running and it's serving files out of the application's directory, but Rails uses its own rules, not the file structure, to decide what gets presented at what URL. For right now, it's enough to know that the name of the controller, hello, will bring up its associated view, which is defined by the *index.html.erb* file.

The initial contents of that file are fairly simple, like those of Example 2-1.

Example 2-1. The default contents of index.html.erb

```
<h1>Hello#index</h1>
<p>Find me in app/views/hello/index.html.erb</p>
```

The Rails designers didn't even give these generated pieces a full HTML document structure. Since the generated code will get replaced anyway, it doesn't matter very much. It's not that Rails doesn't care about the surrounding markup, but rather that the surrounding markup usually comes from layouts, which are covered later in this chapter. If you had generated scaffolding (a larger set of pieces) and not just a view, Rails would also have generated a layout itself. For this chapter's purposes, however, the view is all there is to work with.

For starters, we'll just modify the file a little bit so that it presents a complete HTML document with a slightly friendlier hello, as shown in Example 2-2.

Example 2-2. The new contents of index.html.erb

```
<html>
<head><title>Hello!</title></head>
<body>
<h1>Hello!</h1>
<p>This is a greeting from app/views/hello/index.html.erb</p>
</body>
</html>
```

If you save that file and then reload, you'll see something like Figure 2-2.

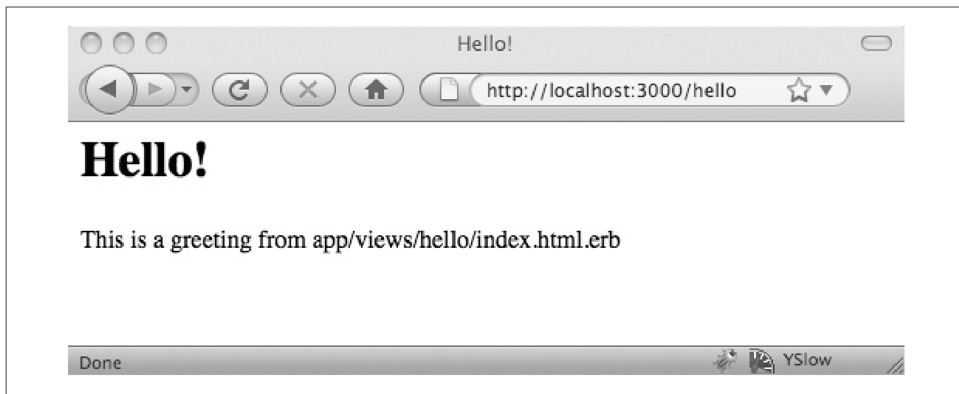


Figure 2-2. A revised greeting

Putting one simple HTML page in the slightly obscure location of a generated HTML page isn't incredibly exciting, but it's a start.

What Are All Those Folders?

The examples in this chapter have called programs in the *script* folder and modified files in the *app* and *public* folders. You might have noticed the large set of folders Rails created for an application. We'll explore most of these in detail over the course of this book, but for now, here's a quick guide to what's there:

app
: Where you build your application's core. It includes subfolders for controllers, helpers, models, and views.

config
: Hosts database configuration, URL routing rules, and the Rails environment structures for development, testing, and deployment.

db
: Provides a home to scripts used to manage relational database tables.

doc
: Collects documentation generated from Ruby code using RubyDoc. RubyDoc is a documentation generator for Ruby, much like JavaDoc. For a lot more information, see *http://www.ruby-doc.org/*.

lib
: Holds code that doesn't quite fit into the model, view, or controller classifications, typically code that's shared by these components or plug-ins you install. The *tasks* subdirectory contains Rake tasks for your application.

log
: Gathers log data—not just errors, but very rich information on requests, how they were processed, how long it took to process them, and session data from the request.

public
: Contains things like stylesheets, images, JavaScript, and things like 404 Not Found error reporting pages.

script
: The home for the prebuilt code you'll be using to generate, run, and interact with large portions of your Rails application.

test
: Contains code—generated at first, but updated by you—for testing your Rails application.

tmp
: Rails' internal home for session variables, temporary files, cached data, etc.

vendor
: Houses plug-ins and gems from outside of Rails itself. Also, if the application has been frozen to a particular version of Rails, that version may be stored here.

Most of the time you'll work in *app* or *test*, with some ventures into *public* to work on the parts of your application (like stylesheets, JavaScript, or images) that Rails doesn't control directly.

Adding Some Data

As pretty much every piece of Rails documentation will suggest, views are really meant to provide users with a perspective on data managed by a controller. It's a little strange to run through all this generation and layers of folders just to create an HTML file. To start taking advantage of a little more of Rails' power, we'll put some data into the controller for *hello*, *hello_controller.rb*, and then incorporate that data into the view.

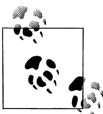
If you open *app/controllers/hello_controller.rb*, you'll see the default code that Rails generated, like that in Example 2-3.

Example 2-3. A very, very basic controller that does nothing

```
class HelloController < ApplicationController

  def index
  end
end
```

This is the first real Ruby code we've encountered, so it's worth explaining a bit. The name of the class, `HelloController`, was created by the script generator based on the name we gave, Hello. Rails chose this name to indicate the name and type of the class, using its normal convention for controllers. Controllers are defined as Ruby classes, which inherit (`<`) most of their functionality from the `ApplicationController` class. (You don't need to know anything about `ApplicationController`s, or even classes—at least not yet—so if you don't understand at this point, just enjoy the generated code and keep reading.)



If you need to learn more about Ruby to be comfortable proceeding, take a look at Appendix A, "An Incredibly Brief Guide to Ruby."

`def index` is the start of the `index` method, which Rails will call by default when it's asked for a Hello. As you can see, it comes to a nearly immediate `end`, which is followed by the `end` for the class as a whole. If we want to make the `index` method do anything, we'll have to add some logic. For our current purposes, that logic can stay extremely simple. Defining a few variables, as shown in Example 2-4, will let us play with the basic interactions between controllers and views, and allow the view to do a few more interesting things. (Example 2-4 is part of the code in *ch02/hello002*.)

Example 2-4. A basic controller that sets some variables

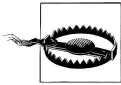
```
class HelloController < ApplicationController

  def index
    @message="Hello!"

    @count=3

    @bonus="This message came from the controller."
  end
end
```

Variables whose names start with `@` are called instance variables. They belong to the class that defines them and have the convenient property of being accessible from the associated view.



When choosing variable names, always be very careful to avoid the enormous list of reserved words presented at *http://wiki.rubyonrails.org/rails/pages/ReservedWords*.

If you use those names, you may find not only that your programs don't run correctly, but also that the supporting development environment misbehaves in strange and annoying ways.

To actually use those variables, make some changes to the view as in Example 2-5.

Example 2-5. Modifying index.html.erb to use instance variables from the controller

```
<html>
<head><title><%= @message%> </title></head>
<body>
<h1><%= @message %></h1>
<p>This is a greeting from app/views/hello/index.html.erb</p>
<p><%= @bonus %></p>
</body>
</html>
```

There are three new pieces here, highlighted in bold. Each contains the name of one of the instance variables from *hello_controller.rb*, surrounded by the `<%=` and `%>` tags. When Rails processes this document, it will replace the `<%= ... %>` with the value inside. You can, of course, create those values from much more complex sources than just a simple variable, but it's easier to see what's happening here in a simple example.