

Extracted from:

Agile Web Development with Rails

Fourth Edition

This PDF file contains pages extracted from *Agile Web Development with Rails*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2010 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The
Pragmatic
Programmers

For
Rails 3.1

Agile Web Development with Rails

Fourth Edition



Sam Ruby
Dave Thomas
David Heinemeier Hansson

*with Leon Breedt, Mike Clark, Justin Gehtland,
James Duncan Davidson, and Andreas Schwarz*

Edited by Susannah Davidson Pfalzer



The Facets  of Ruby Series

Agile Web Development with Rails

Fourth Edition

Sam Ruby
Dave Thomas
David Heinemeier Hansson

with Leon Breedt
Mike Clark
James Duncan Davidson
Justin Gehtland
Andreas Schwarz

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



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 The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

Copyright © 2011 Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-13: 978-1-934356-54-8

Printed on acid-free paper.

Book version: P2.0—August 2011

Ruby on Rails is a framework that makes it easier to develop, deploy, and maintain web applications. During the months that followed its initial release, Rails went from being an unknown toy to being a worldwide phenomenon, and more important, it has become the framework of choice for the implementation of a wide range of so-called Web 2.0 applications.

Why is that?

Rails Simply Feels Right

First, a large number of developers were frustrated with the technologies they were using to create web applications. It didn't seem to matter whether they were using Java, PHP, or .NET—there was a growing sense that their job was just too damn hard. And then, suddenly, along came Rails, and Rails was easier.

But easy on its own doesn't cut it. We're talking about professional developers writing real-world websites. They wanted to feel that the applications they were developing would stand the test of time—that they were designed and implemented using modern, professional techniques. So, these developers dug into Rails and discovered it wasn't just a tool for hacking out sites.

For example, *all* Rails applications are implemented using the Model-View-Controller (MVC) architecture. Java developers are used to frameworks such as Tapestry and Struts, which are based on MVC. But Rails takes MVC further: when you develop in Rails, you start with a working application, there's a place for each piece of code, and all the pieces of your application interact in a standard way.

Professional programmers write tests. And again, Rails delivers. All Rails applications have testing support baked right in. As you add functionality to the code, Rails automatically creates test stubs for that functionality. The framework makes it easy to test applications, and as a result, Rails applications tend to get tested.

Rails applications are written in Ruby, a modern, object-oriented scripting language. Ruby is concise without being unintelligibly terse—you can express ideas naturally and cleanly in Ruby code. This leads to programs that are easy to write and (just as important) are easy to read months later.

Rails takes Ruby to the limit, extending it in novel ways that make a programmer's life easier. This makes our programs shorter and more readable. It also allows us to perform tasks that would normally be done in external configuration files inside the codebase instead. This makes it far easier to see what's happening. The following code defines the model class for a

project. Don't worry about the details for now. Instead, just think about how much information is being expressed in a few lines of code.

```
class Project < ActiveRecord::Base
  belongs_to :portfolio
  has_one    :project_manager
  has_many  :milestones
  has_many  :deliverables, through: :milestones

  validates :name, :description, presence: true
  validates :non_disclosure_agreement, acceptance: true
  validates :short_name, uniqueness: true
end
```

Two other philosophical underpinnings keep Rails code short and readable: DRY and convention over configuration. DRY stands for *don't repeat yourself*: every piece of knowledge in a system should be expressed in just one place. Rails uses the power of Ruby to bring that to life. You'll find very little duplication in a Rails application; you say what you need to say in one place—a place often suggested by the conventions of the MVC architecture—and then move on. For programmers used to other web frameworks, where a simple change to the schema could involve them in half a dozen or more code changes, this was a revelation.

Convention over configuration is crucial, too. It means that Rails has sensible defaults for just about every aspect of knitting together your application. Follow the conventions, and you can write a Rails application using less code than a typical Java web application uses in XML configuration. If you need to override the conventions, Rails makes that easy, too.

Developers coming to Rails found something else, too. Rails isn't playing catch-up with the new de facto web standards; it's helping define them. And Rails makes it easy for developers to integrate features such as Ajax and RESTful interfaces into their code, because support is built in. (And if you're not familiar with Ajax and REST interfaces, never fear—we'll explain them later in the book.)

Developers are worried about deployment too. They found that with Rails you can deploy successive releases of your application to any number of servers with a single command (and roll them back equally easily should the release prove to be somewhat less than perfect).

Rails was extracted from a real-world, commercial application. It turns out that the best way to create a framework is to find the central themes in a specific application and then bottle them up in a generic foundation of code.

When you're developing your Rails application, you're starting with half of a really good application already in place.

But there's something else to Rails—something that's hard to describe. Somehow, it just feels right. Of course, you'll have to take our word for that until you write some Rails applications for yourself (which should be in the next forty-five minutes or so...). That's what this book is all about.

Rails Is Agile

The title of this book is *Agile Web Development with Rails*. You may be surprised to discover that we don't have explicit sections on applying agile practices X, Y, and Z to Rails coding.

The reason is both simple and subtle. Agility is part of the fabric of Rails.

Let's look at the values expressed in the Agile Manifesto as a set of four preferences:¹

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Rails is all about individuals and interactions. There are no heavy toolsets, no complex configurations, and no elaborate processes. There are just small groups of developers, their favorite editors, and chunks of Ruby code. This leads to transparency; what the developers do is reflected immediately in what the customer sees. It's an intrinsically interactive process.

Rails doesn't denounce documentation. Rails makes it trivially easy to create HTML documentation for your entire codebase. But the Rails development process isn't driven by documents. You won't find 500-page specifications at the heart of a Rails project. Instead, you'll find a group of users and developers jointly exploring their need and the possible ways of answering that need. You'll find solutions that change as both the developers and the users become more experienced with the problems they're trying to solve. You'll find a framework that delivers working software early in the development cycle. This software may be rough around the edges, but it lets the users start to get a glimpse of what you'll be delivering.

1. <http://agilemanifesto.org/>. Dave Thomas was one of the seventeen authors of this document.

In this way, Rails encourages customer collaboration. When customers see just how quickly a Rails project can respond to change, they start to trust that the team can deliver what's required, not just what has been requested. Confrontations are replaced by "What if?" sessions.

That's all tied to the idea of being able to respond to change. The strong, almost obsessive, way that Rails honors the DRY principle means that changes to Rails applications impact a lot less code than the same changes would in other frameworks. And since Rails applications are written in Ruby, where concepts can be expressed accurately and concisely, changes tend to be localized and easy to write. The deep emphasis on both unit and functional testing, along with support for test fixtures and stubs during testing, gives developers the safety net they need when making those changes. With a good set of tests in place, changes are less nerve-racking.

Rather than constantly trying to tie Rails processes to the agile principles, we've decided to let the framework speak for itself. As you read through the tutorial chapters, try to imagine yourself developing web applications this way: working alongside your customers and jointly determining priorities and solutions to problems. Then, as you read the more advanced concepts that follow in Part III, see how the underlying structure of Rails can enable you to meet your customers' needs faster and with less ceremony.

One last point about agility and Rails: although it's probably unprofessional to mention this, think how much fun the coding will be.

Who This Book Is For

This book is for programmers looking to build and deploy web-based applications. This includes application programmers who are new to Rails (and perhaps even new to Ruby) and ones who are familiar with the basics but want a more in-depth understanding of Rails.

We presume some familiarity with HTML, Cascading Style Sheets (CSS), and JavaScript, in other words, the ability to view source on web pages. You do not need to be an expert on these subjects; the most you will ever be expected to do is to copy and paste material from the book, all of which can be downloaded.

How To Read This Book

The first part of this book makes sure you are ready. By the time you are done with it, you will have been introduced to Ruby (the language), you will have been exposed to an overview of Rails itself, you will have both Ruby

and Rails installed, and you will have verified this installation with a simple example.

The next part takes you through the concepts behind Rails via an extended example; we build a simple online store. It doesn't take you one by one through each component of Rails ("here is a chapter on models, here is a chapter on views," and so forth). These components are designed to work together, and each chapter in this section tackles a specific set of related tasks that involve a number of these components working together.

Most folks seem to enjoy building the application along with the book. If you don't want to do all that typing, you can cheat and download the source code (a compressed tar archive or a zip file).² This download contains separate sets of source code for Rails 3.0 and Rails 3.1. The files you will want will be in a rails31 directory. See the README-FIRST file for more details.

The third part of the book, starting [on page ?](#), surveys the entire Rails ecosystem. This starts with the functions and facilities of Rails that you will now be familiar with. It then covers a number of key dependencies that the Rails framework makes use of that contribute directly to the overall functionality that the Rails framework delivers. Finally, there is a survey of a number of popular plugins that augment the Rails framework and make Rails an open ecosystem rather than merely a framework.

Along the way, you'll see various conventions we've adopted.

Live Code

Most of the code snippets we show come from full-length, running examples that you can download. To help you find your way, if a code listing can be found in the download, there'll be a bar before the snippet (just like the one here).

Download rails31/work/demo1/app/controllers/say_controller.rb

```
class SayController < ApplicationController
  > def hello
  > end

  def goodbye
  end

end
```

This contains the path to the code within the download. If you're reading the ebook version of this book and your ebook viewer supports hyper-

2. http://pragprog.com/titles/rails4/source_code has the links for the downloads.

links, you can click the bar, and the code should appear in a browser window. Some browsers (such as Safari) will mistakenly try to interpret some of the templates as HTML. If this happens, view the source of the page to see the real source code.

And in some cases involving the modification of an existing file where the lines to be changed may not be immediately obvious, you will also see some helpful little triangles on the left of the lines that you will need to change. Two such lines are indicated in the previous code.

Ruby Tips

Although you need to know Ruby to write Rails applications, we realize that many folks reading this book will be learning both Ruby and Rails at the same time. You will find a (very) brief introduction to the Ruby language in [Chapter 4, Introduction to Ruby, on page ?](#). When we use a Ruby-specific construct for the first time, we'll cross-reference it to that chapter. For example, this paragraph contains a gratuitous use of `:name`, a Ruby symbol. In the margin, you'll see an indication that symbols are explained [on page ?](#).

David Says...

Every now and then you'll come across a *David Says...* sidebar. Here's where David Heinemeier Hansson gives you the real scoop on some particular aspect of Rails—rationales, tricks, recommendations, and more. Because he's the fellow who invented Rails, these are the sections to read if you want to become a Rails pro.

Joe Asks...

Joe, the mythical developer, sometimes pops up to ask questions about stuff we talk about in the text. We answer these questions as we go along.

This book isn't meant to be a reference manual for Rails. Our experience is that reference manuals are not the way most people learn. Instead, we show most of the modules and many of their methods, either by example or narratively in the text, in the context of how these components are used and how they fit together.

Nor do we have hundreds of pages of API listings. There's a good reason for this—you get that documentation whenever you install Rails, and it's guaranteed to be more up-to-date than the material in this book. If you install Rails using RubyGems (which we recommend), simply start the gem documentation server (using the command `gem server`), and you can access all the

Rails APIs by pointing your browser at <http://localhost:8808>. You will find out [on page ?](#) how to build even more documentation and guides.

In addition, you will see that Rails itself helps you by producing responses that clearly identify any error found, as well as traces that tell you not only what point the error was found but how you got there. You can see an example in [Figure 16, *Our application spills its guts.*, on page ?](#). If you need additional information, peek ahead to [Section 10.2, *Iteration E2: Handling Errors*, on page ?](#) to see how to insert logging statements.

Should you get really stuck, there are plenty of online resources to help. In addition to the code listings mentioned, there is a forum,³ where you can ask questions and share experiences; an errata page,⁴ where you can report bugs; and a wiki,⁵ where you can discuss the exercises found throughout the book.

These resources are shared resources. Feel free to post not only questions and problems to the forum and wiki but also any suggestions and answers that you may have to questions others may have posted.

Let's get started! The first steps are to install Ruby and Rails and to verify the installation with a simple demonstration.

3. <http://forums.pragprog.com/forums/148>

4. <http://www.pragprog.com/titles/rails4/errata>

5. <http://www.pragprog.com/wikis/wiki/RailsPlayTime>