Preface

Ruby on Rails is a full-stack framework for developing web applications. Rails embraces many good ideas that are familiar in the Java world: the Model-View-Controller (MVC) pattern, unit testing, agile development, the ActiveRecord pattern, and many others. At the same time, Rails challenges many standard practices: Instead of miles of XML configuration files, Rails relies on conventions where possible. Rails is built with Ruby, a dynamic language, and is deployed as source code.

But forget the technical points for a moment. The reason that any of this matters is that Rails programmers are *getting things done*, and fast. Rails programmers have made (and substantiated) some amazing claims about developer productivity. They are having a lot of fun, too.

Should Java programmers be alarmed by this upstart? Absolutely not. Java programmers are uniquely positioned to take advantage of Ruby on Rails. This book will explain how to get started.

Who Should Read This Book?

This book is for all Java programmers. OK, let us narrow that down a little. This book is for two subsets of Java programmers:

- Those who want to program in Ruby and Rails
- Those who do not

To the first group: We wrote this book because we love Java, and we love Rails. We believe that Java programmers are uniquely qualified to take advantage of Rails, because Java programmers have lived through a lot of the struggles behind the good (and sometimes controversial) ideas in Rails.

To the second group: Rails is not for everything, just like any other tool isn't. However, Rails is such an interesting tool, and Ruby is different from Java in so many fascinating ways, that we think it is the single best complement you can learn to round out your skill set.

To both groups: We have had a great time writing this book, because we share a common language with you, our readers. By assuming a common vocabulary of the Java language and patterns, we are able to move quickly to the meat of topics. We believe that, page for page, this is a much better book for Java developers than a general-purpose book can ever be. Yes, that's bragging, and we are boasting about you, our fellow Java developers. Thanks for all the work you have put in to build a baseline of industry knowledge on which we hope to build.

Why *This* Rails Book?

A lot of Rails books exist. One aspect that sets this book apart is our Java background. We focus on the parts of Rails that will be different, new, and interesting to a Java developer.

The second aspect that sets this book apart is our emphasis on Rails as an ecosystem, not just as a framework. As a Java developer, you are accustomed to having an enormous ecosystem around your programming language. You have great IDEs, monitoring tools, and widgets for every situation. Rails has an ecosystem too—not as big as Java's but important nevertheless. In this book, we spend less time hashing through every random API detail in Rails. Instead, we demonstrate the key points and then move into the ecosystem to show how those key points are used, extended, and sometimes even replaced.

Who Should Read Some Other Book?

This book is a reference for experienced Java programmers who want to learn Ruby and Rails. This is not a tutorial where each chapter walks you through building some sample application. For a tutorial, plus a general introduction to the Ruby language, we recommend *Programming Ruby* [TFH05]. For a tutorial and introduction to Rails, we recommend *Agile Web Development with Rails* [TH06].

This book is not a comparison of Java and Ruby for managers considering a Ruby project. For that, we recommend *From Java to Ruby: Things Every Manager Should Know* [Tat06].

This book is not an introduction for nonprogrammers; for that we recommend *Learn to Program* [Pin06].

Why Ruby on Rails?

Rails is making programmers productive and happy. Plus, we are finding that using Ruby exercises our minds more than any other mainstream language. If you want to start a watercooler conversation about the merits of Ruby and Rails, here are a few talking points:

- *Full-stack web framework*. Rails includes everything you need: Model-View-Controller, O/RM, unit testing, and build and deployment automation. Because everything is tightly integrated, it is ridiculously easy to get started.
- *Opinionated software*. Rails is not designed to let you do anything. It is designed to help you do *the right things*.
- *Convention over configuration*. The danger of both the previous points is that you might not be able to customize the framework to meet your needs. Rails avoids this with convention over configuration. All of Rails' moving parts are held together by convention, but you can override those conventions whenever you need to do so. You get to pay as you go, relying on conventions where necessary and overriding only exactly what you need.
- *One language for application and configuration*. Rails uses Ruby for configuration as well as for application code. Ruby is easier to manage than XML and much more powerful when configuration becomes complex.
- *The secret sauce is Ruby*. Ruby is powerful and elegant, and it has become the language we think in most of the time. Ruby includes good ideas from mainstream programming languages. As a Java programmer, you will have a head start in understanding Ruby's approach to classes, objects, inheritance, and polymorphism. Ruby also includes many features of Smalltalk and Lisp that are missing from mainstream languages. As a Java programmer, you will be delighted to discover how blocks, closures, duck typing, metaprogramming, and functional programming can make your code more expressive and maintainable. Rails is the gateway drug; Ruby is the addiction.

How to Read This Book

All readers should read the entirety of Chapter 1, *Getting Started with Rails*, on page 20. The chapter includes instructions for quickly setting up your environment so you can follow along with all the example code.

Next you have a choice: Ruby first or Rails first? If you are a bottom-up learner who cannot pass by a line of code without understanding it completely, start with the Ruby chapters (Chapter 2, *Programming Ruby*, on page 38 and Chapter 3, *Ruby Eye for the Java Guy*, on page 72). Ruby is radically different from Java, even more than the syntax suggests. Your investment will pay for itself quickly.

If you are the "getting things done" type, jump straight into Rails, which begins with Chapter 4, *Accessing Data with ActiveRecord*, on page 96 and continues through the rest of the book. When you see Ruby idioms that interest you, you can always return to the chapters about the Ruby language. (If you don't know the Ruby name for something, just use Appendix A, on page 303. The dictionary is organized by Java terminology and includes pointers to relevant sections in the book.)

Other than that, feel free to skip around. The book is extensively cross-referenced throughout, so you cannot get too lost.

Make sure you follow the instructions in the next section for downloading the sample code. Ruby and Rails enable an interactive development experience, and you will learn much more if you follow along with the examples.

How to Get Sample Code

The sample code for the book uses Rails version 1.1.6 or newer[1] and Ruby version 1.8.4 or newer. All the sample code for the book is available as a single zip file online.[2]

The sample code includes two Rails applications, named People and Rails XT. The People application is extremely simple and demonstrates how to use Rails to create a front end for a single database table. We build the entire People application from scratch as we go through the book. Section 1.2, *Rails App in Fifteen Minutes*, on page 21 has instructions to set up the People application.

Rails XT stands for "Rails Exploration Testing." The Rails XT application doesn't have a unified feature set that addresses some problem domain. Instead, Rails XT is a holding tank for dozens of fragments that

1. A few examples rely on features in Rails 1.2, which is still under development as of this writing. These examples are noted in the text as they occur.
2. See http://pragmaticprogrammer.com/titles/fr_rails4java/code.html

demonstrate Rails' capabilities. Because of its heterogeneous nature, Rails XT requires a bit more setup. You don't need to set up Rails XT to get started. When you need to do so, you can find instructions in the sidebar on page 98. Here is a quick overview of the sample directory structure:

rails_xt
: This contains the Rails exploration tests (see Section 1.6, *Running the Unit Tests*, on page 32) and the Quips sample application. Throughout the book, Ruby examples should be executed from this directory unless otherwise noted.

java_xt
: You will use the Java exploration tests throughout the book.

appfuse_people
: You will use the Java People application throughout the book.

junit4
: You will find any tests that require JUnit4 here.

Rake
: This includes Rake and Ant examples from Chapter 8, *Automating the Development Process*, on page 233.

hibernate_examples
: This includes Hibernate examples from Chapter 4, *Accessing Data with ActiveRecord*, on page 96.

The Java examples are split into several directories to simplify classpath management. That way, you can install just the libraries you need. For example, you don't need to install Struts, Hibernate, and so on, to run the language examples in java_xt.

How We Developed the Java Examples

This is a book about two worlds: the world of Java programming and the world of Rails programming. Whenever worlds collide, you can expect to hear statements like "Java sucks, and Rails rocks..." (or the reverse).

You won't hear that tone here. To us, that is like a carpenter saying "Hammers suck, and saws rock." Carpenters use many tools, and programmers should too. More important, the confrontational approach limits an important opportunity. When you have multiple ways to solve a problem, you can learn a lot by comparing them.

Our goal in visiting this new world (Rails) is to learn by comparison with our shared history (Java). But what exactly is our shared history? Ruby on Rails is a web framework, which means you could compare it to about a gazillion things in the Java world. Should we look at Java? Plain servlets? Servlets plus JSP? Aged MVC frameworks such as Struts? Rich component frameworks such as Tapestry? Java EE standard architectures such as JSF? Or all of these?

When we needed a Java baseline to compare with Rails, we chose Struts, Hibernate, and Axis. We picked these because our careful statistical research indicated these were best-known among Java developers. Moreover, we limit our Java usage to techniques that are typical in applications we have seen in the field. As a result, the Java code in this book should look familiar to most Java web developers.

(We asked a lot of people.)

The downside of this approach is that "typical" and "familiar" Java code is not necessarily best practice. So although this approach is useful for teaching Rails, it does not provide a comprehensive review of Java best practices. (That's a whole 'nother book.) Where we have skipped interesting Java approaches for reasons of space, we have included margin notes and references at the ends of the chapters.

Many of the Java examples are built starting with Matt Raible's excellent AppFuse (http://www.appfuse.org). AppFuse is a metaframework that allows you to quickly jump-start a web application using the frameworks of your choice. If you want to compare Rails to Java frameworks not covered in this book, AppFuse is a great place to start.

Acknowledgments

We would like to thank our wives. Joey and Lisa, none of this would have happened, or would have meant as much, without you. We would also like to thank our extended families. Without your love and support, this book would have been stalled until at least 2025.

Thanks to our reviewers: David Bock, Ed Burns, Scott Davis, Mark Richards, Ian Roughley, Brian Sletten, Venkat Subramaniam, Bruce Tate, and Glenn Vanderburg. We would never have believed that such a talented, busy group of people could devote so much time and attention to this project. Thank you; this book is immeasurably better for it.

To the Pragmatic Programmers: Thank you for building the kind of publishing company that can produce a book like this, on this timeline. You are consummate professionals.

To the Relevance Gang: We are in for an exciting ride. Thanks for your smarts, thanks for your excellent work, but thanks most for the fun environment.

To the Pragmatic Studio: Thanks for leading the way in getting Ruby and Rails people together, all over the country. We can't wait for the first Rails Edge.

To the No Fluff, Just Stuff Gang: Thanks for sharing our secret lives. Our ideas about Java (and Ruby) are sharpened every weekend at our semiclandestine encounters.

To Jay Zimmerman: Thanks for building a community around excellent people and around excellence in software development.

To James Duncan Davidson: Thanks for spreading the Mac meme.

To Neal Ford: Thanks for the cross-the-board expertise, from agility and DSLs all the way to food and fashion. Who says we have to specialize?

To Bruce Tate: Thanks for helping kick-start our Rails consulting business and for being a companion in our professional journey. You were country when country wasn't cool.

To Dave Thomas: You make everything around you better, and you have fun doing it. Thanks for your inestimable contributions to Ruby, to Rails, and to our work.

To Jim Weirich: Thanks for the just-in-time technical support on FlexMock questions.

To Al von Ruff: Thanks for your work on the Internet Speculative Fiction Database.[3] We have enjoyed it as readers, and we particularly appreciate you making the schema and data available for some of the examples in this book.

To Matt Raible: Thanks for AppFuse. Without it we'd still be in a bottomless pit of XML configuration files.

To the folks at Coke, Pepsi, Red Bull, Macallan, and Lagavulin: Thank you for the beverages that fueled this book. Bet you can't guess which drinks go with which chapters!

Yes, we drink both Coke and Pepsi. And we like both Java and Ruby.

3. http://www.isfdb.org