

Peter Armstrong

Foreword by Stuart Eccles

FLEXIBLE RAILS

FLEX 3 ON RAILS 2

SAMPLE CHAPTER

 MANNING





Flexible Rails
by Peter Armstrong
Chapter 1

Copyright 2008 Manning Publications

brief contents

PART 1	GETTING STARTED	1
	1 ■ Why are we here? Where are we going?	3
	2 ■ Hello World	14
	3 ■ Getting started	52
PART 2	BUILDING THE APPLICATION.....	103
	4 ■ Creating the main Flex UI	105
	5 ■ Expanding the Rails code, RESTfully	118
	6 ■ Flex on Rails	186
	7 ■ Validation	261
PART 3	REFACTORING.....	293
	8 ■ Refactoring to Cairngorm	295
	9 ■ Holding state on the client properly	369
PART 4	FINISHING UP.....	419
	10 ■ Finishing the application	421
	11 ■ Refactoring to RubyAMF	468
	12 ■ Rails on AIR (Adobe Integrated Runtime)	512

Part 1

Getting started

In this part, we'll do the necessary setup work to get to the fun stuff in the rest of the book. We'll install everything, do a Flex and Rails version of "Hello World," and then get user creation and login working in Rails and hook up the Flex UI to it.

This part contains three iterations:

- *Iteration 1: "Why are we here? Where are we going?"*—This iteration provides the motivation for the book, an understanding of the history of Flex and Rails and how they fit together, and an overview of the book.
- *Iteration 2: "Hello World"*—This iteration contains three separate sets of instructions (Windows or Mac OS X + Flex Builder 3, Windows + Flex SDK, and Mac OS X + Flex SDK) for installing everything we need and getting "Hello World" running. You only need to read the section that applies to you.
- *Iteration 3: "Getting started"*—In this iteration, we'll set up MySQL and add account creation and login functionality to our Rails application, using the `restful_authentication` plugin. We'll then hook up the Flex UI to use the Rails account creation and login functionality. Finally, we'll set up the most minimal of tests. At the end of this iteration, we'll have a good starting point for any Flex + Rails application.

Why are we here? Where are we going?

*HTML sucks all the joy out of programming for me
HTML+CSS, that is
I'm so glad I don't have to do the design work for our apps
I'm trying to design a simple form
and I'm hating life
It's seriously making me want to not work on this anymore
...
html makes it so easy to write forms that look like crap
and SO HARD to write forms that look nice
that's so backwards*

—Jamis Buck,
Signal vs. Noise [Fly on the Wall], July 17, 2007¹

¹ <http://www.37signals.com/svn/posts/495-fly-on-the-wall-paying-attention-to-users-mow-the-lawn-vs-cut-the-grass-chowder-html-forms>.

There is a *lot* of hype these days around Flex and Rails. I'll try my hand at it for a few paragraphs, too.

Ruby on Rails, or just Rails for short, has been revolutionizing web application development since its introduction in 2004. Nowadays, it seems that a new “Web 2.0” company that uses Rails is spawned every 10 seconds.

Flex is a sexy framework that lets us write code that feels more like coding a desktop application—except it runs inside the Flash player! Because it targets the Flash player, we can build new Rich Internet Applications (RIAs) without worrying about browser compatibility nonsense, JavaScript, CSS, and so on.

NOTE The preferred term now seems to be *rich Internet applications*. I don't prefer it, though, because rIa isn't a good-looking acronym. As a curmudgeonly form of protest (I'm an old-school Flex developer—I used to code Flex 1.0 while walking uphill both ways in the snow...), I'm going to call them Rich Internet Applications in the book. Also, the full capitalization of Rich Internet Applications may be coming back into fashion, in response to a Microsoft evangelist having attempted to make RIA stand for “Rich Interactive Applications”—so I'm shouting “get off my lawn” in an avant-garde way, I guess.

Because Flex 3 targets one platform (Flash 9), we don't have to worry about platform compatibility issues. The Write Once, Run Anywhere (WORA) dream that client-side Java programmers had—before it turned into “write once, debug everywhere”—can finally be realized, but with Flex. Flex achieves what previous technologies such as Java applets failed miserably in attempting: applications that feel like desktop applications, but which run inside any modern web browser on Windows and Mac.

NOTE Write Once, Run Anywhere was essentially realized on the server side but not on the client side. On the client side, AWT was terrible, and Swing doesn't look like any of the platforms it runs on. SWT is an excellent alternative to Swing, because it gives us native widgets. However, SWT can't be used in an applet yet, so we can't run it in a web browser. It's just useful for building applications like Eclipse—and like Flex Builder 3, which is built on top of the Eclipse Rich Client Platform (RCP).

But here's a little-known secret, which this book is the first book to cover: *Flex and Rails work amazingly well together!*

We can use Flex 3 and Rails 2 to build RIAs today that look and feel more like Web 3.0 than many of the “me too [point oh]” Web 2.0 sites we see copying each other today. This book will show you how to get started doing exactly this.

In this iteration, we'll get an overview of Flex and Rails, their history, and how they can be used together.

Iterations

In this book, the chapters are called *iterations*. I've done this because we'll develop an application iteratively throughout the book—it has nothing specific to do with Flex or Rails. (That said, both Flex and Rails lend themselves to an iterative style of development.) Each iteration advances the state of the application further. You can start following along with the book at the beginning of any iteration, using the code from the end of the previous iteration.

This chapter has no code—it's just an introduction. I'm calling it "iteration 1" instead of "introduction" as a cunning way of getting you to read it, because many people skip introductions and dive right into chapter 1. *I love it when a plan comes together!*

1.1 Overview of the features and strengths of Flex 3 and Rails 2

Now that you're all excited, let's take a deep breath and get an overview of both platforms. This section will present a high-level overview of both and then show how they can be combined. Don't worry if you don't understand a particular point here; it will be explained later.

1.1.1 Overview of Flex 3

In Flex 3, we write code in MXML (XML files with a .mxml extension; *M* for Macromedia) and ActionScript (text files with a .as extension) files and compile them into a SWF file, which runs in the Flash player. This SWF is referenced by an HTML file, so that when a user with a modern web browser loads the HTML file, it plays the Flash movie (prompting the user to download Flash 9 if it's not present). The SWF contained in the web page can interact with the web page it's contained in and with the server it was sent from.

Even if you've never created a Flash movie in your life, don't consider yourself a designer, and wouldn't recognize the Timeline if you tripped over it, you can use Flex to create attractive applications that run in the Flash player. Flex development is easily learned by any intermediate-level developer with either web or desktop UI (such as Windows Forms or Java Swing) programming experience.

1.1.2 Overview of Rails 2

Figure 1.1 shows how Rails provides a standard three-tier architecture (presentation tier, model tier, persistence tier) as well as a Model-View-Controller (MVC) architecture. As the diagram shows, Rails takes care of everything between the web server and the database.

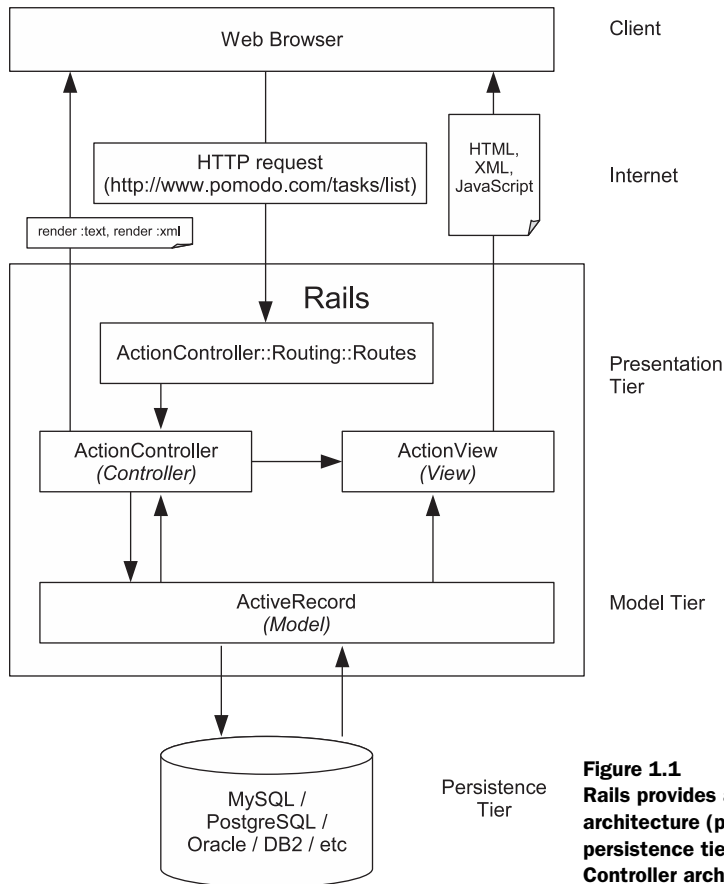
ITERATION 1*Why are we here? Where are we going?*

Figure 1.1
Rails provides a standard three-tier architecture (presentation tier, model tier, persistence tier) as well as a Model-View-Controller architecture.

The typical sequence is as follows:

- 1 A user visits a particular URL in their web browser (makes an HTTP request).
- 2 This request goes over the Internet to the web server in which Rails is running (such as WEBrick, lighttpd, Mongrel, or Apache).
- 3 That web server passes the request to the routing code in Rails, specifically `ActionController::Routing::Routes`. These routes are defined in `config/routes.rb`. The default route turns HTTP requests into method calls on controllers.
- 4 The controller (such as `TasksController`) method (such as `index`) is called. It communicates with various ActiveRecord models (which are persisted to and retrieved from a database of our choosing). The controller method then can do one of the following things:

- Set some instance variables and allow a view template (a specially named .html.erb file, for example) to be used to produce HTML, XML, or JavaScript, which is sent to the browser. This is the job of Action View. Together, Action View and Action Controller form Action Pack.
- Bypass the view mechanism and do rendering directly via a call to the render method. This method can produce plain text (`render :text => "foo"`), XML (`render :xml => @task`), and so on.

1.1.3 Overview of using Flex 3 and Rails 2 together

Figure 1.2² shows how Flex and Rails can be used together.

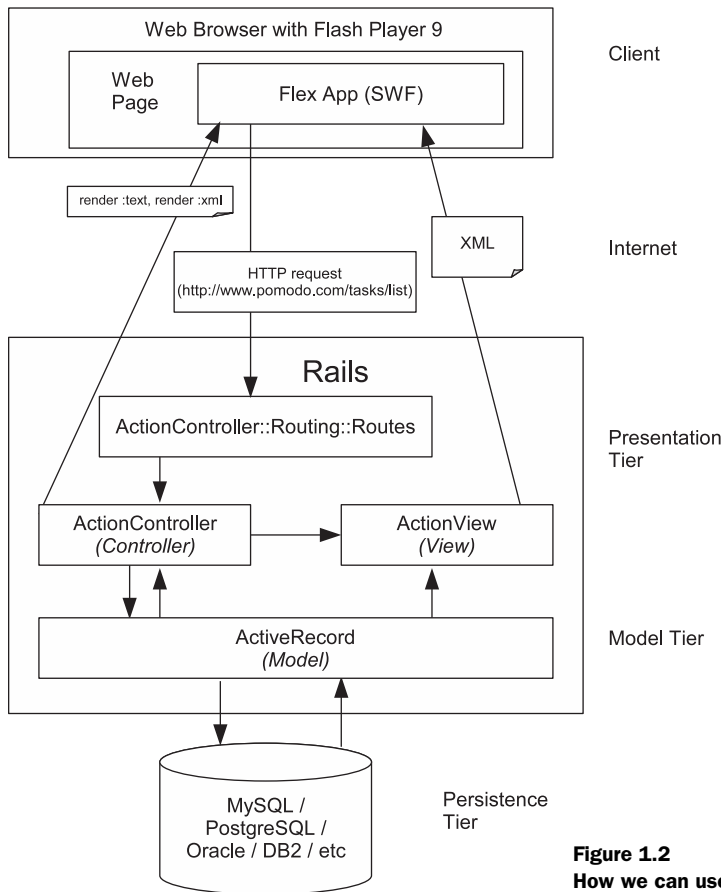


Figure 1.2
How we can use Flex and Rails together

² The diagram “Fig 7. The complete MVC Architecture for a Thin-Client Web Application” at <http://www.uidesign.net/Articles/Papers/UsingMVCPatterninWebInter.html> inspired the design of my block diagrams.

1.2 Flash 9? Are you kidding me?

The reference to Flash 9 earlier may have set off alarm bells in your head: “Isn’t Flash 9 somewhat new? How many people will be able to run my app?” Table 1.1 should put this concern in perspective³.

Table 1.1 Worldwide ubiquity of Adobe Flash Player by version—June 2007

	Flash Player 7 (released September 2003)	Flash Player 8 (released August 2005)	Flash Player 9 (released June 2006)
Mature markets	99.3%	98.5%	90.3%
US/Canada	99.4%	98.7%	90.5%
Europe	99.1%	98.2%	90.5%

As of June 2007, Flash 9 has over 90% market penetration in “mature markets” (US, Canada, UK, Germany, France, and Japan). Furthermore, note that Flash 8 has achieved 98% market penetration in less than two years—which is extremely good.

Despite how productive Flex 3 and Rails 2 are for development, it will still take you *some* time to build your killer app. And in that time, our target market is getting larger by the day. (If you haven’t accomplished much in a given day, you can still feel good that you grew your target market.)

Finally, note that most of your early adopters will be, well, early adopter types. These are the TechCrunch reading, Digg/del.icio.us/reddit using types. These people will have Flash 9 or won’t mind getting it.

One more thing: If you work in an enterprise environment, the adoption percentages for different Flash versions among consumers today are much less of a concern than if you’re trying to develop a consumer-facing product. As long as your IT department allows the Flash player to be installed, you can mandate that users upgrade their Flash Player versions when they first use your app. The installation and upgrade process is extremely smooth, which is a major reason why you see Flash used everywhere today, whereas Java applets are little more than a historical curiosity.

³ This is taken from a much more complete set of tables at http://www.adobe.com/products/player_census/flashplayer/version_penetration.html.

NOTE You might think this is a significant hurdle (and it may be one in your case), but note that AJAX apps have their own security issues because of cross-site scripting, and so on—IT departments sometimes have issues with JavaScript as well. At least with Flex 3, it’s a binary decision: If our user has or can get Flash 9, you’re good to go. With AJAX, it’s a question of IE 6, IE 7, Firefox 1.0, Firefox 1.5, Safari, Opera, JavaScript enabling, ad infinitum (definitely doable, but by no means as simple).

Speaking of history, it’s nice to know a bit of the history of Flex and Rails, to see how they have evolved over the last few years. This is useful because it helps us understand why no one was thinking about using Flex with Rails in 2004, why a few people started thinking about it in 2006, and why many people are thinking of the combination now.

1.3 History

In 2004, two frameworks were released that have gone on to dominate web and RIA development: Macromedia Flex in March 2004 and Ruby on Rails in July 2004. The two frameworks initially couldn’t have seemed more different.

Ruby on Rails was a free, Open Source, web application framework that strongly appealed to web developers who were frustrated either with PHP or with J2EE. IDEs were spurned, and a fairly obscure Macintosh-only text editor called TextMate was hailed as the greatest achievement of Western civilization. (Emacs and vi were for old people, presumably.) In many blog postings, the enterprise was portrayed as something evil to be ignored, changed, or destroyed. XML was to be avoided at all costs: In *Agile Web Development with Rails* (Dave Thomas et al, Pragmatic Bookshelf, 2006), reason #10 of “Dave’s Top 10 Reasons To Like Rails” is “No XML!” The claim was often made that we could write an entire web application in Rails in fewer lines of code (LOC) than the amount of code just in the XML configuration files of a web application built with EJBs.

NOTE For an example of an article highlighting the reduced LOC, see http://rewrite.rickbradley.com/pages/moving_to_rails/.

Regarding XML, Ruby 1.8 (which Rails runs on) does have support for XML, but Rails prefers to use YAML for its configuration files. (YAML, which stands for YAML Ain’t Markup Language, is a “straightforward machine parsable data serialization format designed for human readability and interaction with scripting languages”: www.yaml.org.) The phrase “No XML!” can be rephrased more accurately but less catchily as “No XML configuration files everywhere, and no XML needed to define our database schema.”

Macromedia Flex 1.0 and 1.5 (which both used ActionScript 2.0) were server products that ran in a J2EE application server that compiled MXML files and ActionScript files into Flash applications (SWFs). Typically, MXML files were used to lay out GUI components, which were developed in either MXML or ActionScript. Flex 1.0 and 1.5 were priced at enterprise levels: about \$15,000 USD *per CPU* for the server product.⁴ Because the server side of a Flex application was typically J2EE, a lot of XML configuration files were typically needed along with XML for the database mapping.

MXML files can, and typically do, contain ActionScript in inline `<mx:Script>` blocks. MXML files are transformed into ActionScript before being compiled into a SWF along with the ActionScript files, so the “what should be done in ActionScript and what should be done in MXML” line was always blurry. A large project typically has lots of both kinds of files. To summarize, table 1.2 shows what the two frameworks looked like in their early days.

Table 1.2 Flex 1.0 and Rails 1.0 compared

	Flex 1.0	Rails 1.0
Cost	Expensive!	Free
Code is	Proprietary	Open source
XML is	Everywhere	Considered evil
IDE	Flex Builder (based on Dreamweaver)	None

In the more than three years since Rails’ release, a lot has changed. Rails has become one of the most influential frameworks in web application development. It seems that every day, some new Web 2.0 app is released that is built on it. This is partly due to the marketing prowess of David Heinemeier Hansson (DHH to his followers) and 37signals, but also to a large extent due to the productivity advantages in faster development time and reduced lines of code to maintain that Rails provides. Rails has also vastly improved its support for XML: ActiveRecord now has a `to_xml` method that we’ll use a lot in this book.

In the past three years, Flex has also progressed rapidly for an enterprise-class product, as shown in table 1.3.

Flex 3 has better performance and download sizes (due to the framework cache) than Flex 2. Furthermore, Flex 2 has *vastly* better performance (in some

⁴ This wasn’t competing with PHP; it was competing with Laszlo, which also was a very expensive product that let you write GUI code in XML that got compiled to a SWF.

Table 1.3 Flex version history

Date	Flex version	ActionScript version
March 2004	1.0	2.0
October 2004	1.5	2.0
June 2006	2.0	3.0
October 2007	3.0 Beta 2	3.0

case, up to *ten times* faster) than Flex 1.0 and 1.5, as well as better XML handling and an updated version of ActionScript. Another advantage of Flex 2 over Flex 1.0 and 1.5 is cost savings: Whereas Flex 1.0 and 1.5 are expensive server products, Flex 2 and Flex 3 can be used in their Flex Framework SDK versions with a command-line compiler without paying Adobe a penny. Flex 3 also features a further cost savings for IDE users: Flex Builder 3 Standard Edition is half the price of Flex Builder 2. (Of course, if you want the new profiler, you're still paying a nontrivial sum to get the Professional Edition.)

In April 2007, Adobe announced the open sourcing of the Flex 3 framework under the Mozilla Public License (MPL). This is huge news in the world of RIA development.

NOTE Although the Flex framework has been open sourced, the Flash player is not open source. This, of course, prompts the typical reaction among the more vocal free software advocates. My position is that the open sourcing of Flex *is* big news and that the choice of the MPL (as opposed to, say, the GPL) is a huge step by Adobe in ensuring the commercial adoption of Flex.

That said, even though the Flex framework is free and is being open sourced, if you're using Windows or Mac you may want to buy Flex Builder 3. It will sell⁵ for \$249 USD (Standard Edition) or \$699 USD (Professional Edition, which will include the charting components, profiler, and so on).

NOTE Adobe also sells a server-side product called Live Cycle Data Services (formerly Flex Data Services), which has a restricted free version for smaller deployments. It won't be covered in this book.

⁵ <http://www.onflex.org/ted/2007/10/flex-3-beta-2-lower-price-flex-builder.php>.

Table 1.4 shows what the two frameworks look like now.

Table 1.4 Flex 3 and Rails 2 compared

	Flex 3.0	Rails 2.0
Cost	Free	Free
Code is	Open Source	Open Source
XML is	Everywhere	Ambivalent; it's still avoided in configuration files, but XML output is included by default in RESTful controllers
IDE	Flex Builder 3 (\$249 Standard Edition, \$699 Professional Edition)	Free (Aptana RadRails, Eclipse with the RDT plug-in, NetBeans) and commercial offerings

Flex 3 is much more similar to Flex 2 than Flex 2 is to Flex 1.5. If it wasn't for the addition of the Adobe Integrated Runtime (AIR) which we'll introduce in iteration 12, Flex 3 probably should have been called Flex 2.5. (The code in this book was originally written in Flex 2; it compiled and ran in Flex 3 unchanged.)

Now that we understand how Flex and Rails have evolved and where they are today, let's look at what we'll accomplish in this book.

1.4 A preview of the book

The project we'll create throughout this book is called *pomodo*. Why pomodo? Because it's a stupid, meaningless name, and a prominent feature of Web 2.0 is meaningless names, often with missing vowels. (It also features rounded corners and gradient fills,⁶ which we will use, too.) Pomodo will be a variation on a To Do list application.

What will be different about pomodo? For one, its UI will be in Flex, so we can create a cool-looking To Do list with little effort. Second, it will be a *Getting Things Done* (GTD) style To Do list, meaning that tasks will be organized into projects and will have locations. In addition, pomodo will use the concept of a Next Action, which is essentially the next task in each project that has nothing blocking it.

Why a To Do list application? There are two reasons. First, because of 37signals' Ta-da List, Basecamp, and Backpack products, the Rails community used to seem a bit obsessed with To Do lists: In the early days of Rails, they were often the "one step beyond Hello World" application built in many fine tutorials

⁶ It turns out that Stuart Eccles makes this same joke in his presentation to the London Flash Platform User Group: <http://www.lfpug.com/ruby-on-rails-for-the-flex-developer-22062006-stuart-eccles/#more-11>.

GTD

David Allen's *Getting Things Done* (Penguin, 2002) is a great book about time management. If you haven't read it already, buy it and read it. Briefly, it involves going to Staples and spending \$500 on office supplies and a big filing cabinet (I did this), writing down everything in your brain (a la the main character in the film *Memento*, who had to always remind himself, "Remember Sammy Jenkins"), and using lots of folders (43 of them—hence the name of the popular website www.43folders.com) to run your life. The elegance and efficacy of the approach, combined with the paper cuts and environmental devastation caused by using so much paper, has inspired countless programmers (myself included) to write their own GTD-style application or to abuse an existing tool (such as a wiki, outlining tool, Outlook, or Gmail) to make it work in a GTD style. The problem with GTD as presented in David Allen's book is that it assumes our world revolves around paper (and, quaintly, that we have a secretary). This may be true for the executives who go to his seminars, but it's *not* true for software developers.

online. Second, and more seriously, since the application is a GTD-style To Do list application, it will have enough features to demonstrate a significant subset of Flex and Rails features, but still be small enough and with a simple enough domain to be fully understood while learning the frameworks and how they interact. I could have created something (say, a cool-looking chess game) that better showed off the eye-candy features of Flex. However, I decided this wouldn't have been as useful: Most of us are (unfortunately) building applications that look more like To Do lists than games. Also, since this is a book about how Flex and Rails can be used together, the pure eye-candy features are superfluous.

1.5 Summary

Flex and Rails fit together well and have evolved to be a better fit because of Flex going Open Source. We can develop with Flex Builder using either Windows or OS X, or with the Flex SDK using Windows, OS X, or Linux.



FLEXIBLE RAILS

Peter Armstrong

Foreword by Stuart Eccles, Made by Many, Ltd.

Flex on Rails may sound like an odd coupling but, in fact, the two are a great fit. Adobe's once expensive and proprietary (now free and open-source) rich user interface platform (Flex) used on top of the upstart, open-source, Ruby-driven web development environment (Rails) is like a Ferrari automobile—beautiful on the outside, fast on the inside, and incredible to drive. Flex offers amazing new UI options to go with Rails' famously-short development cycles.

Flexible Rails teaches Flex 3 on Rails 2 techniques in an unusually effective way: its chapters take an application through eleven iterations, each increasingly “real world.” You'll learn how Flex and Rails integrate with HTTPService and XML and see how Rails' RESTful controller design supports both Flex and HTML clients. Along the way, you'll learn to use the Cairngorm MVC framework in large Flex applications and work with the RubyAMF Flash Remoting gateway.

Written with ongoing feedback from over one thousand Early Access readers, **Flexible Rails** is practical, code-focused, and readable. It is accessible to web or desktop UI developers who have “hello world” level exposure to Rails and little or no knowledge of Flex.

What's Inside

- All content based on Flex 3 and Rails 2
- A complete, professional-quality (and MIT-licensed) Flex on Rails application
- Rails on AIR, the Adobe Integrated Runtime

Peter Armstrong is a Vancouver-based RIA developer who uses Flex and Rails daily. He is a frequent public speaker and was a key part of the team that won the 2006 Adobe MAX Award for RIA/Web Development.

For more information, code samples, and to purchase an ebook visit www.manning.com/FlexibleRails

“... a first-class learning experience that is tough to quantify but easy to qualify: Don't miss it.”

—Louis F. Springer
Sun Microsystems

“*Flexible Rails* changes the face of web applications development.”

—Robert Dempsey
Atlantic Dominion Solutions

“Well done! ... a thorough job of blending two of the best technologies!”

—Christopher Bailey
Bring Light

“Flex 3 on Rails 2 is much more than the sum of its parts—and this book is the key.”

—Arne Pfeilsticker
Pfeilsticker Software
+ Services GmbH

“A really nice book that covers all about Flex and Rails working together.”

—Paulo Fernando Larini
Benner Health Systems

