Extracted from:

# The Rails View

## Creating a Beautiful and Maintainable User Experience



The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

# The Rails View

## Create a Beautiful and Maintainable User Experience

John Athayde
and Bruce Williams

*Edited by Brian P. Hogan*

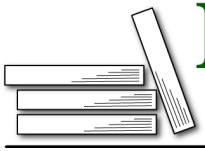
The Facets of Ruby Series

# The Rails View

## Creating a Beautiful and Maintainable User Experience

John Athayde
Bruce Williams

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf



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*.

The team that produced this book includes:

Brian Hogan (editor)
Potomac Indexing, LLC (indexer)
Molly McBeath (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)



Printed in the United States of America.
ISBN-13: 978-1-93435-687-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2012

# Preface

In 2004, Rails was born and the web discovered the MVC (model-view-controller) pattern in earnest, which brought a whole new level of productivity and *fun* to a world of developers and designers.

You'll find no end of books that provide a firm foundation for writing controllers and models (which benefit greatly from being written top-to-bottom in plain Ruby), but when it comes to views—that meeting place of Ruby, HTML, JavaScript, and CSS (not to mention developers and designers)—what's a disciplined craftsman to do?

This book aims to widen the discussion of Rails best practices to include solid, objective principles we can follow when building and refactoring views. By the time you're finished reading, you'll understand how you can structure your front end to be less brittle and more effective and boost your team's productivity.

## Taming the Wild West

For all the advantages that Rails has over traditional, everything-in-the-view approaches like vanilla PHP or ASP, it's also fostered a culture of complacency around how views are structured and maintained.

After all, with all the controller and model logic extracted and the addition of helpers, what could go wrong?

While many of the elements that comprise the view are seen as easy (HTML, for example), the view layer in its entirety is an incredibly complex thing. This complexity can be so daunting that developers and designers just give up and use tables, hackery, and any tweak they can just to make it look somewhat right on the front end.

There are a lot of reasons for this. Many developers are uneasy around the view layer, being in such a hurry to get out of it and back to "real code" that they slap things together and leave a mess. Technical debt in the view layer

often goes unpaid, and knowledge of good markup practices can be years behind or even considered irrelevant. After all, it works all right!

Designers can be uneasy around generated code and, without training, see ERB blocks as a sort of magical wonderland they can't hope to understand. Helpers are just black boxes, and the underlying model relationships and controller context that drive our views are just as opaque. Many designers are so visually focused that they, too, disregard the importance and usefulness of correct, modern markup. After all, it looks all right!

It's easy for the view layer to become a no-man's-land that no one owns or adequately polices or a junkyard that no one feels safe to walk through.

In this book we'll work hard to convince you not to abdicate responsibility for the view layer. We'll work together to learn how we can build application views sustainably from the ground up, discover useful refactoring patterns and helpful tools, and tackle integrating disparate technologies like Ruby, HTML, and JavaScript into a cohesive unit that's more than just a stumbling block between you and the new features you need to implement.

## Who Should Read This Book?

If you're a designer working with Rails or a Rails developer working in the view layer, this book is for you. We'll cover the technical issues present in the view layer, and we'll also highlight some unique challenges that mixed teams of developers and designers face when working together.

## Ruby and Rails Versions

*The Rails View* was built on top of Rails 3.2.1 and Ruby 1.9.3 and should be compatible with future stable releases for quite some time. In the event that we have small compatibility issues with future versions, we will post updates in the online forum on the book's website.[1]

Much of the content and code would need to be modified to work with some earlier versions due to our coverage of the Rails 3.1+ asset pipeline and use of the new Ruby 1.9 Hash literal syntax.

You can check your Rails version with the following command:

```
% rails -v
```

1. http://www.pragprog.com/titles/warv/

You can use gem install with the -v option to manually get the appropriate version.

```
% gem install rails -v 3.2.1
```

To manage your Ruby versions, we recommend RVM (Ruby Version Manager).[2]

## What Is in the Book?

We'll learn how to build solid, maintainable views in Rails over the next nine chapters.

In Chapter 1, *Creating an Application Layout*, on page ?, we look at how to build the view structure for a new application from the ground up and get our layout files in order to provide a firm foundation for the rest of our application.

In Chapter 2, *Improving Readability*, on page ?, we look at how we can make our templates easier to read and more naturally convey their intent.

In Chapter 3, *Adding Cascading Style Sheets*, on page ?, we'll introduce you to the asset pipeline, explain the new SCSS format, customize the Sprockets configuration, and talk about how we can package assets into reusable units.

In Chapter 4, *Adding JavaScript*, on page ?, we'll continue our discussion of the asset pipeline, highlighting CoffeeScript, the Rails UJS drivers, and some organizational techniques for including JavaScript plugins in our applications.

In Chapter 5, *Building Maintainable Forms*, on page ?, we tackle forms, investigate creating our own form builders, and use some existing libraries to make complex forms easier to build and maintain.

In Chapter 6, *Using Presenters*, on page ?, we learn some techniques to make displaying complex information as easy and maintainable as possible from the view, building abstractions with our own custom Ruby classes.

In Chapter 7, *Handling Mobile Views*, on page ?, we discuss the challenges we face with supporting different screen resolutions and geometries, including mobile devices, and what solutions exist to aid in reusing templates and styling or whether to separate them altogether.

2. http://rvm.beginrescueend.com

In Chapter 8, *Working with Email*, on page ?, we discover some tips and tricks to make sending rich email less frustrating and designing emails less dependent on trial-and-error.

Finally, in Chapter 9, *Optimizing Performance*, on page ?, we'll learn the basics of measuring and solving application and business performance problems.

## How to Read This Book

Each chapter in this book builds upon the content in the previous chapter. While examples will center around the ArtFlow application that we'll begin to build in Chapter 1, *Creating an Application Layout*, on page ?, chapters can be read sequentially or by jumping around to focus on a specific problem. You should be able to pull the code from our repository for any given chapter and work with it.

Chapter 1, *Creating an Application Layout*, on page ?, covers a lot of HTML and CSS that may seem out of place for a Rails book, but we feel these topics are critical to writing good views. Spend some time refreshing yourself on this subject matter even if you are already familiar with it. You may find some surprises in there!

## Online Resources

The book's website has links to an interactive discussion forum as well as to errata for the book.[3] You'll also find the source code for all the projects we built. Readers of the ebook can click the gray box above the code excerpts to download that snippet directly.

If you find a mistake, please create an entry on the errata page so we can address it. If you have an electronic copy of this book, use the links in the footer of each page to easily submit errata to us.

Let's get started by looking at how views work and by digging into how we deliver those to our application's visitors.

3. http://www.pragprog.com/titles/warv/