

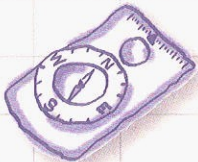
A Brain-Friendly Guide

Head First Rails

**A Learner's Companion
to Ruby on Rails**



Master your
data with
Rails finders

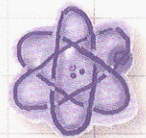


Integrate your apps
with Google Maps and
never get lost again

Learn how Suzy
validated all
her dates and
avoided another
awful evening



Get inside Embedded
Ruby, and take
control of your apps



Add Ajax and
XML to your Rails
universe

O'REILLY®

David Griffiths

Table of Contents (Summary)

	Intro	xxi
1	Really Rapid Rails: <i>Getting Started</i>	1
2	Rails Apps, Made to Order: <i>Beyond Scaffolding</i>	45
3	Everything Changes: <i>Inserting, Updating, and Deleting</i>	103
4	Truth or Consequences?: <i>Database Finders</i>	153
5	Preventing Mistakes: <i>Validating Your Data</i>	187
6	Bringing It All Together: <i>Making Connections</i>	219
7	Avoiding the Traffic: <i>Ajax</i>	263
8	It All Looks Different Now... : <i>XML and Multiple Representations</i>	307
9	Taking Things Further: <i>REST and Ajax</i>	357
10	Rails in the Real World: <i>Real-World Applications</i>	401

Table of Contents (the real thing)

Intro

Your brain on Rails. Here *you* are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing Rails?

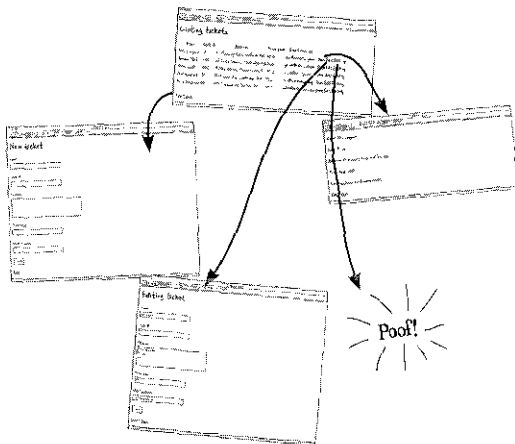
Who is this book for?	xxii
We know what you're thinking	xxiii
Metacognition	xxv
Bend your brain into submission	xxvii
Read me	xxviii
The technical review team	xxx
Acknowledgments	xxxi

getting started

1

Really Rapid Rails

Want to get your web app development off to a flying start? Then you need to know **Rails**. Rails is the **coolest** and **quickest** **development framework** in town, allowing you to develop **fully functional web apps** quicker than you ever thought possible. Getting started is simple; all you need to do is **install Rails**, and start turning the pages. Before you know it, you'll be **miles ahead of the competition**.



The application needs to do lots of things	3
So what things do we need for the app?	4
Rails is for database-centric apps like the ticket sales system	6
You create a new application with the rails command	7
Now you need to add your own code to the default app	9
Scaffolding is generated code	10
There are no tables in the database yet!	14
Create the table by running a migration	15
Sweet! You saved your buddy's job!	19
To modify an app, you need to dig into the app's architecture	20
The 3 parts of your app: model, view, and controller	21
Rails Exposed	22
The 3 types of code are kept in separate folders	25
The files in the view need to be edited	26
Edit the HTML in the view	27
The application needs to store more information now	31
A migration is just a Ruby script	32
Rails can generate migrations	33
Give your migration a "smart" name, and Rails writes your code for you	34
You need to run your migration with rake	35
But changing the database isn't enough	36

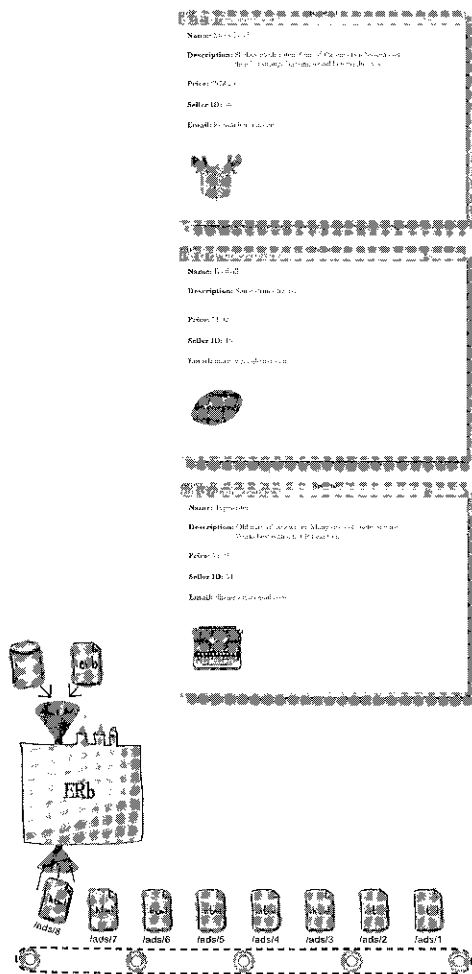
beyond scaffolding

Rails apps, made to order

2

So what's really going on with Rails? You've seen how **scaffolding** generates heaps of code and helps you write web applications wicked fast, but what if you want something a little different? In this chapter you'll see how to really **seize control** of your Rails development and take a look underneath the hood of the framework. You'll learn how Rails decides which **code** to run, how **data** is read from the database, and how **web pages** are generated. By the end, you'll be able to publish data the way **you** want.

Scaffolding does way too much	49
Let's start by generating the eBay model...	50
... and then we'll actually create the table using rake	51
But what about the controller?	52
The view is created with a page template	54
The page template contains HTML	55
A route tells Rails where your web page is	57
The view doesn't have the data to display	64
So what should the page show?	65
The controller sends the ad to the view	66
Rails turned the record into an object	68
The data's in memory, and the web page can see it	69
There's a problem - people can't find the pages they want	73
Routes run in priority order	76
To get data into the view, you will also need code in the controller	78
An index page will need data from all of the records	79
Ad.find(:all) reads the whole table at once	80
The data is returned as an object called an array	81
An array is a numbered sequence of objects	82
Read all of the ads with a for loop	86
We need HTML for each element in the array	87
Rails converts page templates into Ruby code	88
Loops can be added to page templates using scriptlets	89
On each pass of the loop, the page generates one link	90
So what does the generated HTML look like?	91
But there are two page templates...	
should we change the code of each one?	94
But what about the new static content eBay sent over?	97

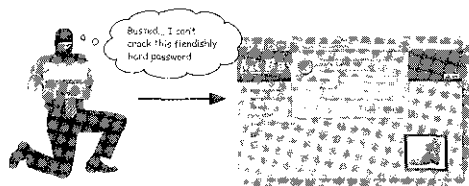
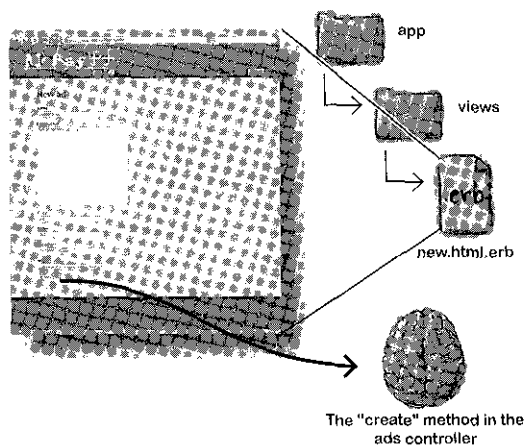


inserting, updating, and deleting

Everything changes

3

Change is a fact of life—especially for data. So far you've seen how to whip up a quick Rails application with scaffolding, and how to write your own code to publish data from a database. But what if you want users to be able to edit data *your way*? What if scaffolding doesn't do what you want? In this chapter, you'll learn how to **insert**, **update**, and **delete** data in exactly the way you want. And while you're doing that, you'll be taken deeper into how Rails *really* works and maybe even learn a little about security along the way.



People want to post new ads online	104
You already know how to build an app that publishes data from the database	105
Saving data works just the opposite of reading data	106
You need a form to submit data and an action method to save it	107
Are forms and objects related?	109
Rails can create forms that are associated with model objects	110
The @ad form object has not been created	114
The form object needs to be created before the form is displayed	115
The forms ad object will be created in the new action of the controller	116
Each page template now has a matching controller method	117
The form doesn't send an object back, it sends data back	119
Rails needs to convert the data into an object before it can be saved	120
The controller create method, step-by-step	121
The controller needs to save the record	122
Don't create a new page, use an existing one	128
But how can a controller action display another action's page?	129
Redirects let the controller specify which view is displayed	130
But what if an ad needs to be amended after it's been posted?	133
Updating an ad is just like creating one... only different	134
Instead of creating an ad, you need to find one; instead of saving it, you need to update the ad	135
Restricting access to a function	142
... but now old ads need to be deleted	145
Doing it yourself gave you the power to do more than scaffolding	151

4

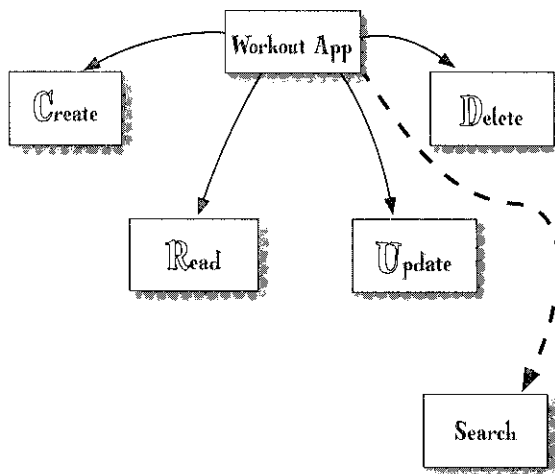
Truth or consequences?

Every decision you make has consequences.

In Rails, knowing how to make **good decisions** can save you both time and effort. In this chapter, we'll look at how **user requirements** affect the choices you make, right from the very **beginning** of your app. Should you use scaffolding and modify the generated code? Should you create things from scratch? Either way, when it comes time to customize your app further, you need to learn about **finders**: *getting at your data* in a way that makes sense to you and serves your **users' needs**.

Keep fit with the Rubyville Health Club	154
The application actually looks pretty close...	157
We're going to fix the scaffolding	158
Design the search function	159
Let's start by building the form	160
Add the search to the interface	163
How do we find client records?	171
We only need those records where client_name = the search string	172
There's a finder for every attribute	173
We need to match either the client name or the trainer name	178
Finders write database queries	179
We need to be able to modify the conditions used in the SQL query	180
Use :conditions to supply SQL	181

Business is really taking off but we're having trouble keeping track of all the personal fitness sessions of our clients. Think you can help?



validating your data

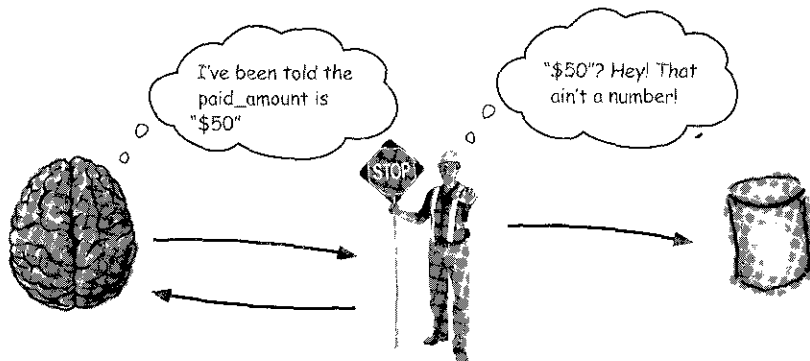
5

Preventing mistakes

Everyone makes mistakes... but many of them are preventable!

Even with the very best of intentions, your users will still enter bad data into your web app, **leaving you to deal with the consequences**. But just imagine if there was some way of **preventing mistakes** from happening in the first place. That's where **validators** come in. Keep reading, and we'll show you how to add **clever Rails validation** to your web app so that you can **take control** of what data is allowed in—and what needs to be kept out.

Watch out—there's bad data in the room	188
Validation code goes in the MODEL	190
Rails uses validators for simple validation	191
So how do validators work?	192
Let's check if something is a number	194
Users have been leaving out data on their workout forms	196
How do we check for mandatory fields?	197
Validators are simple and work well	200
Something strange has happened at McBay	203
The validators work, but they don't display errors	204
If you build your own pages, you need to write your own error message code	207
The controller needs to know if there was an error	208
We still need to display error messages!	212
The McBay system is looking pretty sweet	214



making connections

Bringing it all together

6

Some things are stronger together than apart.

So far you've had a taste of some of the **key Rails ingredients**. You've created entire web applications and taken what Rails generates and **customized** it for your needs. But out in the real world, **life can be more complex**. Read on.. it's time to build some **multi-functional web pages**. Not only that, it's time to deal with **difficult data relationships**, and take control of your data by writing your own **custom validators**.



Coconut Airways need a booking system	220
We need to see flights and seat bookings together	222
Let's look at what the seat scaffolding gives us	223
We need the booking form and seat list on the flight page	224
How can we split a page's content up into separate files?	225
ERb will assemble our pages	229
So how do we create the booking form partial?	230
Now we need to include the partial in the template	231
We need to give the partial a seat!	234
You can pass local variables to a partial	235
We also need a partial for the seat list	242
People are ending up on the wrong flights	244
A relationship connects models together	245
But how do we define the relationship?	247
But some people have too much baggage	249
We need to write our own validation	250
We need the reverse relationship	253
The system's taken off at Coconut Airways	260

Dude... I booked a flight to a beach party but wound up on a historical tour of an old leper colony!



ajax

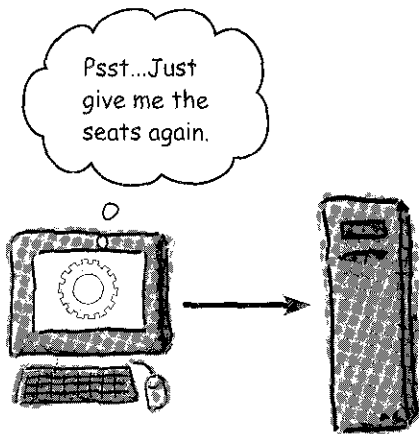
7

Avoiding the traffic

People want the best experiences out of life... and their apps.

No matter how good you are at Rails, there are times when traditional web apps just don't cut it. Sometimes users want something that's more **dynamic** and that responds to their every whim. Ajax allows you to build **fast, responsive web apps**, designed to give your users the **best experience the web has to offer**, and Rails comes complete with its own set of Ajax libraries just waiting for you to use. It's time to **quickly and easily add Ajax goodness** to your web app and please even more users than before.

There's a new offer at Coconut Airways	264
Which parts of a page change most?	265
Doesn't the browser always update the entire page?	270
So what ELSE can make a request?	271
First we need to include the Ajax libraries...	272
...then we need to add an Ajax "Refresh" link	273
The browser <i>needs to ask for an update</i>	278
But should we make the browser ask over and over again?	279
You listen to a timer like you listen to a button	280
Ajax Exposed	284
Someone's having trouble with their bachelor party	285
The form needs to make an Ajax request	286
The form needs to be under the control of JavaScript	287
We need to replace the <i>create</i> method	289
So what effect does this code have?	290
There's a problem with the flight bookings	295
We only know how to update one part of the page at a time	296
The controller needs to return JavaScript instead of HTML	297
So what does Rails generate?	301
If you don't say where to put the response, it will be executed	302



XML and multiple representations

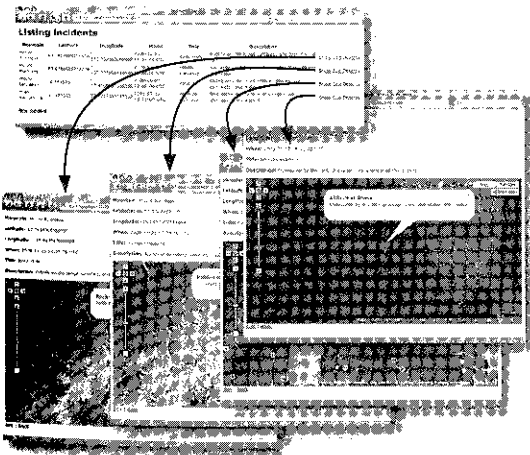
8

It all looks different now...

You can't please everyone all of the time. Or can you?

So far we've looked at how you can use Rails to quickly and easily develop web apps that **perfectly fit one set of requirements**. But what do you do when **other requirements come along**? What should you do if some people want **basic web pages**, others want a **Google mashup**, and yet more want your app available as an **RSS feed**? In this chapter you'll create **multiple representations** of the same basic data, giving you the **maximum flexibility** with **minimum effort**.

Climbing all over the world	308
The users hate the interface!	309
The data needs to be on a map	310
We need to create a new action	311
The new action seems to work...	312
The new page needs a map... that's the point!	313
So what code do we need?	314
The code will only work for localhost	315
Now we need the map data	316
What do we need to generate?	318
We'll generate XML from the model	319
A model object can generate XML	320
What will the controller code look like	321
Meanwhile, at 20,000 feet...	326
We need to generate XML and HTML	327
XML and HTML are just representations	329
How should we decide which format to use?	330
How does the map page work?	334
The code is ready to go live	336
RSS feeds are just XML	344
We'll create an action called news	345
We have to change the structure of the XML	348
So we'll use a new kind of template: an XML builder	349
Now let's add the feed to the pages	353
On top of the world!	355



REST and Ajax

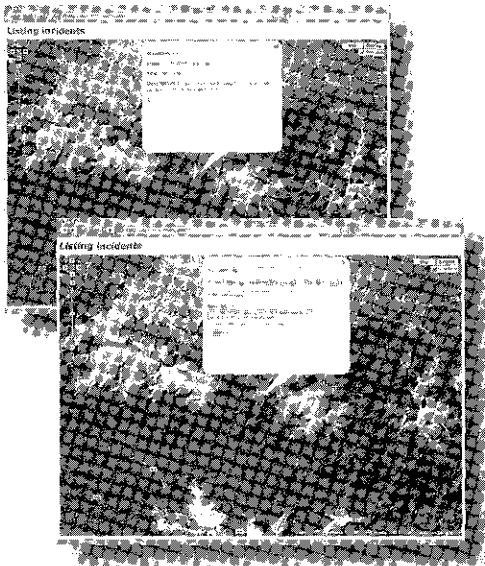
Taking things further

9

It's time to consolidate your mash-up skills.

So far you've seen how you can add **Google Maps** to your web apps to clearly show spatial data. But what if you want to **extend the functionality that's already there**? Keep reading, and we'll show you how you can add more **advanced Ajax goodness** to your **mash-ups**. And what's more, you'll learn a bit more about **REST** along the way.

Too many incidents!	358
The map could show more details	359
We can extend the map using Ajax	360
But how do we convert the index page?	361
What will the "show" action need to generate?	362
The new map functionality is a success!	367
We need to create requests using Ajax, too	368
The map partial lets us specify a "new" action	370
How do we prove an incident was saved?	375
The form needs to update the contents of the pop-up's <div>	376
Avalanche!	381
How things works now..	382
We could have an "Edit" link in the pop-up	383
We'll start by modifying the "edit" action	384
And we'll also need a new link on the show page	386
So how do we use the link_to helper?	387
Ajax links to the rescue	391
We're using the wrong route!	393
The HTTP method affects the route that's chosen	394
So what's an HTTP method?	395
Head First Climbers needs you!	398



10

real-world applications

Rails in the real world

You've learned a lot about Ruby on Rails.

But to apply your knowledge to **the real world**, there are a number of things you need to think about. How do you connect your application to **another database**?

How do you **test** Rails apps? How do you make the most out of the Rails and the **Ruby language**? And where do you find out the latest on **what's happening** with Rails?

Keep reading, and we'll put you on the inside track that will take your development skills to the next level.

Look! It's a big Ruby "Try this" page	405
Web apps need testing too	406
So what kinds of tests are available?	407
Going live	408
So how do you change the database?	409
What's REST?	410
The web application that went astray	411
Living on the Edge	412
Getting more information	413
A little light reading...	414
Head First books on related topics	415
Leaving town...	417

