Extracted from:

Rails for .NET Developers

This PDF file contains pages extracted from Rails for .NET Developers, 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 © 2008 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.

1.6 Instant Gratification—Your First Rails App

The best way to "get it" when starting out with Rails is to just go for it by building a simple application and playing around with its capabilities. In this section, that is exactly what we'll do. We'll build a basic application that's going to help us keep track of all the books (yes, paper!) in our library.

We're going to write almost the entire application using just a few simple commands. You'll have to see it to believe it, so let's fire up a command prompt, cd to where you'd like your application to live (we've chosen the C:\dev directory), and create a new Rails application:

```
C:\dev> rails book_tracker
C:\dev> cd book_tracker
```

The rails command results in the default Rails directory structure being created. Like File > New Project in Visual Studio, it creates a shell of an application for us. Let's go ahead and list the directory contents to see exactly what was created:

```
C:\> dir
app
components
config
db
doc
lib
log
public
script
test
tmp
vendor
```

We'll go into the Rails directory structure in further detail later, but for now, let's focus on the two directories where we'll likely spend the most time. The app directory contains all your main application code, broken down into four subfolders: controllers, helpers, models, and views. And the config directory contains our application's configuration. Among other things, this application configuration describes how we'll connect to our database.

Once we've created the Rails project, it's time for some Rails magic.

CLICK HERE to purchase this book now.

Scaffolding—An App in One Line

Scaffolding, in Rails terms, is a lot like real-world scaffolding—it is boilerplate code to help keep our application in place while we're building the real production-quality code behind it. We can put it up very quickly, and when we're done, we should tear it down so that it doesn't get in the way. Let's put the scaffolding up now:

```
C:\dev\book_tracker> ruby script/generate scaffold book title:string
 author:string on_loan:boolean
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/books
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  app/views/books/index.html.erb
      create  app/views/books/show.html.erb
      create  app/views/books/new.html.erb
      create  app/views/books/edit.html.erb
      create  app/views/layouts/books.html.erb
      create  public/stylesheets/scaffold.css
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/book.rb
      create    test/unit/book_test.rb
      create    test/fixtures/books.yml
      create    db/migrate
      create    db/migrate/20080722191828_create_books.rb
      create  app/controllers/books_controller.rb
      create  test/functional/books_controller_test.rb
      create  app/helpers/books_helper.rb
       route  map.resources :books
```

The generate command, in its simplest form, takes two parameters, the first being what you'd like to generate—in this case a scaffold—and the second being the name of the new class. By convention, the scaffold generator expects the singular form of the resource you're trying to create, so we've passed in the singular book argument to the command.

In addition, we've also told the generator about what fields a book has. We've told it that each book will have two string fields, title and author, and a boolean field that indicates whether we've lent that book to a friend.

CLICK HERE to purchase this book now.

All the scaffold generator command (or any generator command, for that matter) does is create a bunch of files for us. We could have created these files ourselves, but this is a whole lot easier! We'll get into what all these files are for in a moment, but for now, let's concentrate on the db/migrate/20080722191828_create_books.rb file. This file is what's known as a *migration*.

Versioning the Database with Migrations

A *migration* is a Ruby script that uses a very simple *domain-specific language* (DSL) for manipulating databases. As .NET developers, we might be accustomed to inventing our own ways of creating and versioning database schemas. If we were developing an app with .NET and SQL Server, a simple example might go something like this:

1. When developing the first cut of your application, create an initial database creation script by using SQL or by using a graphical tool such as SQL Management Studio and then dumping the schema to a text file.

2. While developing, make changes to the database schema through a similar process—using various SQL scripts—sharing throughout with team members so they can keep up-to-date.

3. After our application is deployed to staging or production environments, use additional SQL scripts we've created to keep the database schemas on your development environment in sync with these other environments in our IT infrastructure.

Migrations are simply the Rails way of doing the same thing. Except that instead of using SQL, it's all written in Ruby. This approach has a couple of benefits:

- Since our app and database manipulation are all written in the same language, there's very little context switching or deep knowledge about database-specific intricacies necessary to be successful building your app.

- Rails does all the low-level SQL for you, and it is completely platform-agostic. We can easily switch from MySQL to Postgres to SQL Server if we want and even have a different database platform per environment. Imagine developing on a Mac with SQLite 3, staging to a Linux box with MySQL, and deploying to a Windows server with SQL Server for production (not that we recommend this, but doing so would be trivial).

CLICK HERE to purchase this book now.

Let's open db/migrate/20080722191828_create_books.rb and take a peek at what a migration looks like:

Ruby

Download instant/20080722191828_create_books.rb

```ruby
class CreateBooks < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.string :title
      t.string :author
      t.boolean :on_loan
      t.timestamps
    end
  end

  def self.down
    drop_table :books
  end
end
```

The fields we passed as parameters to the scaffold generator command are already in this migration file. We are free to modify them in any way at this point; the actual changes to the database schema have not been made yet.

The name of the file is crucial—well, at least the number at the beginning is. This is a time stamp of when the migration was created, and it represents the database version. As we add more migrations to our application, that number will increase; that's how Rails knows the order in which to execute them.

A migration class has two methods: self.up and self.down. The self.up method tells Rails what to do when migrating up; likewise, the self.down method gets executed when rolling the database back.

Remember, the generate command creates a bunch of files—nothing else. The actual database manipulation doesn't take place until we execute the migration file. Let's do that now:

```
C:\dev\book_tracker> rake db:migrate
(in c:/dev/book_tracker)

== 20080722191828 CreateBooks: migrating ==============================
-- create_table(:books)
   -> 0.2267s
== 20080722191828 CreateBooks: migrated (0.2269s) =====================
```

rake is Ruby's automation and task-running utility (more about rake in Chapter 12, *Finishing Touches*, on page 234), and we've just used it

CLICK HERE to purchase this book now.

to run the db:migrate task, thus creating our books table. Wait, where? Remember that, unless you specify otherwise, Rails will use the SQLite 3 engine for its development and test databases by default. If you now list the contents of the db directory, you'll see that your development database (the development.sqlite3 file) has just been created.

Fire It Up

Believe it or not, a complete application that we can use to maintain our book collection is now ready to use! But first, we'll want to start up WEBrick, the lightweight web server that comes with the default Rails installation.[11] WEBrick is the Rails equivalent of Web Developer Server for us ASP.NET devs—it's a technology that lets us run our app locally while developing. And, like Web Developer Server, we access it via localhost on a high-numbered port that doesn't interfere with other services on our machine. WEBrick starts up on port 3000 by default:

```
C:\dev\book_tracker> ruby script\server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
```

Head over to http://localhost:3000 using your web browser of choice. You should see the standard Rails welcome screen, as shown in Figure 1.1, on the following page. Now, simply add the name of the resource you're interested in—in this case books—to the end of that URL so that you end up with http://localhost:3000/books. Then you can marvel at what you've accomplished with one simple command.

As you browse, you'll find that everything you need to create, update, read, and delete books has been automatically generated for you. Not only is the application perfectly usable, but the code that's been generated is a great starting point for understanding how a Rails application is supposed to be built.

Time to Tweak

OK, that was easy. Let's take it a step further now. Let's say we'd now like to capture the date on which each book was purchased.

11. If you have the mongrel gem installed, it will be used instead of WEBrick.

CLICK HERE to purchase this book now.

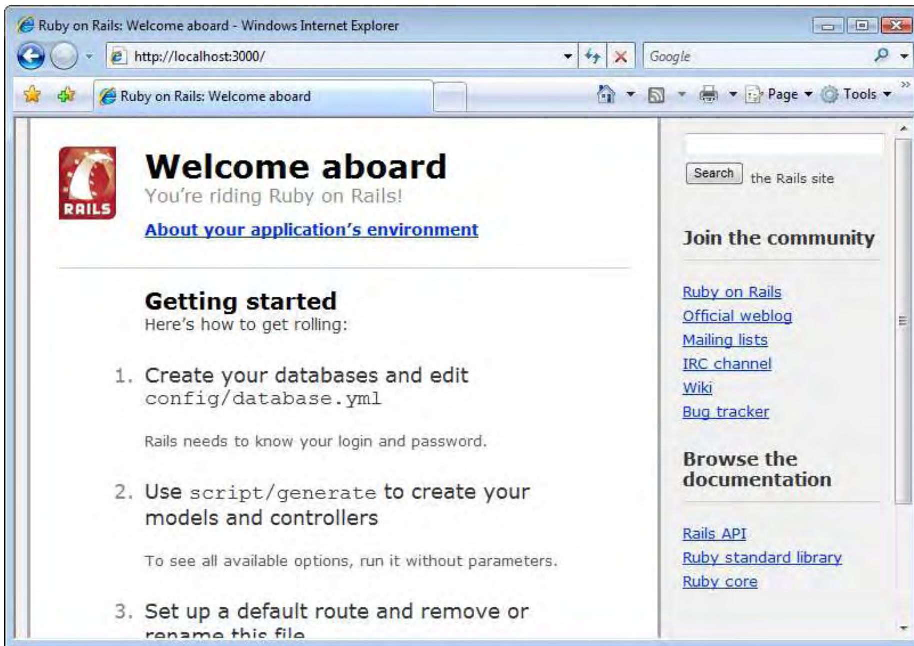

Figure 1.1: Rails welcome screen

Thanks to migrations, adding the field to the database is easy. Simply generate a new migration with this command:

```
ruby script\generate migration add_purchased_on_to_books
    exists  db/migrate
    create  db/migrate/20080722191930_add_purchased_on_to_books.rb
```

This will create a new migration file at db/migrate/20080722191930_add_purchased_on_to_books.rb. Now we'll add code to the self.up method to indicate what should happen when we upgrade from the current version (version 20080722191828) to the next version (version 20080722-191930—as indicated by the first part of the migration's filename). And, we'll also add code to the self.down method in case we ever want to roll back to the previous version.

CLICK HERE to purchase this book now.

Joe Asks...
How Do I Roll Back My Database to a Previous Version?

Migrations can also be undone:

```
C:\dev\book_tracker> rake db:rollback
```

In addition, the rake task used to migrate up also accepts an optional parameter with the target version. For example, the following command would also migrate the database back to version 20080722191828:

```
C:\dev\book_tracker> rake db:migrate VERSION=20080722191828
```

Ruby

Download instant/20080722191930_add_purchased_on_to_books.rb

```ruby
class AddPurchasedOnToBooks < ActiveRecord::Migration
  def self.up
    add_column :books, :purchased_on, :date
  end

  def self.down
    remove_column :books, :purchased_on
  end
end
```

Now, we'll execute the migration:

```
C:\dev\book_tracker> rake db:migrate
(in C:/dev/book_tracker)
== 20080722191930 AddPurchasedOnToBooks: migrating ====================
-- add_column(:books, :purchased_on, :date)
   -> 0.0470s
== 20080722191930 AddPurchasedOnToBooks: migrated (0.0470s) ===========
```

Now that we've added the new field to the database, the last step is to add the new field to the pages used to create, edit, and display books. Open app/views/new.html.erb, and you'll notice that the scaffolding created code that generates text fields for the title and author fields, as well as a checkbox for the "on loan" flag. The methods text_field and check_box in this code are known as *form helpers*. These helper methods tell Rails to show text input and checkbox input HTML tags, respectively, when the page is rendered. Since we've added a date column to the database, we could use a text input field if we wanted, but in most cases, the date_select helper method to display the date input using three drop-down boxes is a more attractive choice.

CLICK HERE to purchase this book now.

Ruby Download instant/new.html.erb

```
<h1>New book</h1>

<%= error_messages_for :book %>

<% form_for(@book) do |f| %>
  <p>
    <b>Title</b><br />
    <%= f.text_field :title %>
  </p>

  <p>
    <b>Author</b><br />
    <%= f.text_field :author %>
  </p>

  <p>
    <b>On loan</b><br />
    <%= f.check_box :on_loan %>
  </p>

  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>

<%= link_to 'Back', books_path %>
```

Head on over to http://localhost:3000/books again, and give it a try. Upon visiting the "create book" page, you should see something similar to what's shown in Figure 1.2, on the next page. Using the same technique, you should also go ahead and enhance app/views/new.html.erb in the same way.

More One-Liners—Validating Input

As we've seen so far, Rails is the king of the one-liners. These simple commands represent a lot of the little things you need to turn your app from a set of simple forms into a full-blown web application.

Try to create a new book with no title and no author. You'll find that there's no validation preventing that from happening. Luckily, there's an easy way to fix that. Open app/models/book.rb, and add a new line of code, just after the class definition:

Ruby Download instant/book.rb

```
class Book < ActiveRecord::Base
        validates_presence_of :title, :author
end
```

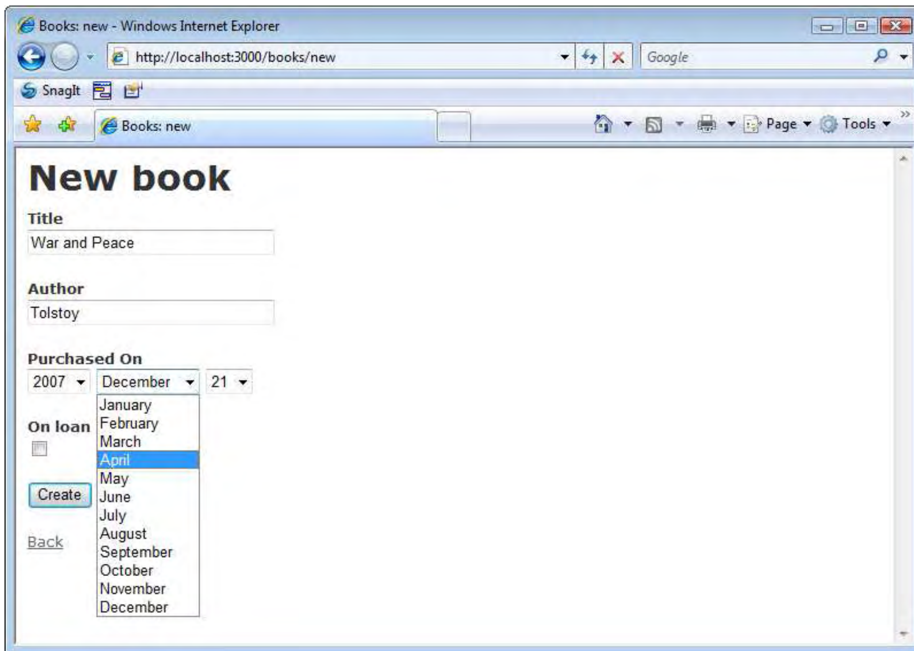

Figure 1.2: Creating a book with a "Purchased On" date

Try to create a new book with no title or author one more time. Voila! Now Rails catches that error and displays an appropriate error message on the page.

In this chapter, we've gotten a solid Rails development stack installed, and we've seen how easy it is to get a simple application up and running. We've already built a real application that talks to a database, learned how to do basic database versioning, and even done some simple form validation. That's not bad for just a few lines of code.

We also built it all without the help of an IDE, which is probably different from what you're used to, if you've been living in the .NET world for a while.

You're now prepared to go a lot deeper into the anatomy of a Rails application and understand how the Ruby language plays a big part in what makes Rails what it is.

CLICK HERE to purchase this book now.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Rails for .NET Developers' Home Page
http://pragprog.com/titles/cerailn
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/cerailn.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragprog.com/catalog
Customer Service:	orders@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com