All aboard! An introduction to Rails 3

Skill Level: Intermediate

Martin Streicher (martin.streicher@gmail.com)
Software Developer
Pixel, Byte, and Comma

23 Mar 2010

The impending release of Ruby on Rails version 3 both refines and expands the capabilities of the popular Web application framework. Offering cleaner controllers and savvier SQL queries, you can expect to write less code than before. Better yet, you can include most of the components of Rails 3 in any Ruby application. Here's a look at what's changed for the better.

Over the past two years, the Ruby on Rails application framework has garnered a cottage industry of hosting and service providers, an expansive and impressive array of development tools, and a wide variety of complementary libraries—called *gems* and *plug-ins* in Ruby parlance—that increase the capabilities of the software. For example, Engine Yard and Heroku are just two companies that provide virtual and headache-free Rails software hosting; the colorfully named Oink and Bullet profile memory usage and performance, respectively; and Clearance and Sunspot provide off-the-shelf authentication and fast, indexed search.

Frequently used acronyms

- **CRUD:** Create, read, update, delete
- **HTML:** Hypertext Markup Language
- **REST:** Representational State Transfer
- **SQL:** Structured Query Language

Since 2007, the Rails community has grown, too. The global legion of Rails developers is vibrant, helpful, and always eager to improve the software. Indeed, it's not hyperbole to say that the community is determined to improve Rails, with coders constantly leapfrogging one another to build something better. Iteration after

iteration, features quickly evolve from nascent to utilitarian to powerful to elegant and ultimately to indispensable. In many instances, gems and plug-ins the community finds essential are enshrined in the Rails core. Rails's *named scopes*—a query shorthand—followed that very trajectory, as did *nested forms,* a fairly recent addition that supplanted previous attempts to create and edit multiple models within the same HTML form. Indeed, perhaps the most difficult task for Rails developers is keeping pace with change. (Luckily, a number of weekly Ruby and Ruby on Rails podcasts organize and present trends and best practices.)

Rails version 3, the next major release of Rails, continues the rapid advancement of the toolkit. True to its heritage, the software remains "opinionated," preferring convention over configuration. Rails's core components—RESTful routes, relationships, validations, templates, and database abstractions—persist. However, much of the internals of those units have been rewritten or refined. Most notably, and borrowing heavily from Merb, many of Rails's essential features are no longer tightly coupled. For instance, the data-validation conveniences previously available only to a Rails application are now stand-alone components and can be included in vanilla Ruby code. Controller capabilities, such as rendering partials and templates, are also independent and can be embedded in any library.

In this article, you'll take a look at Rails 3 and its many changes and additions and create a new Rails 3 application from scratch. As of the middle of February 2010, Rails 3 is a beta prerelease, and the core team is collecting patches, feedback, and documentation to prepare an official release prior to the start of summer. Nonetheless, the current incarnation of Rails 3 is sufficient for building applications and learning about the multitude of new capabilities.

Big changes, little changes

The number of changes in Rails 3 is too significant to recount fully here. To read a complete list along with supplemental material, consult the Rails 3 Release Notes (see Resources for a link). Here, however, are some of the alterations most likely to affect developers:

- **One command to rule them all.** Rails 3 obsolesces the family of scripts (script/server, script/generate, and the rest) found in each application and replaces it with a single command, aptly named `rails`. For example, where you previously typed `./script/console`, you now type `rails console`. The `rails` command also generates new applications, as it did before. Its operation differs according to whether it's launched amid an existing Rails application.
- **A manifest solution for dependencies.** Reconciling and resolving gem dependencies is something of a knotty problem. Gem revisions can vary from one system to another, as can the collection of available gems. With

such variety, it can be difficult to widely deploy or share a Rails application. Rails 3 introduces the Bundler, a special utility for managing dependencies (thus obsolescing `config.gem`). You declare dependences in a catalog named *Gemfile* in the root of your application. The Bundler downloads and stores all the named gems. You can even "pack" the gems in your application to preclude downloads from external repositories.

- **Queries without the queries.** Historically, Rails had made good use of domain-specific languages (DSLs) throughout—think of `has_one` or `validates_numericality_of`—with one notable exception: database queries. Certainly, Rails's dynamic finders ease the burden, but code littered with option hashes replete with `:conditions`, `:order`, and `:limit` are common, as are `find_by_sql` statements. Rails 3 incorporates *relational algebra,* a DSL designed to express queries. Primitives include `project` (to select columns), `where` (to express conditions), `join` (to specify relationships), `take` and `skip` (for limits and offsets, respectively), and `group` (for aggregation), among others.
- **Controllers sans that fussy boilerplate code.** The core actions of a Rails controller—`new`, `create`, `edit`, `update`—typically do not vary, especially if the controller is largely there for CRUD operations. In fact, the output of the controller generator, `./script/generate controller`, often suffices without further modification. Given those similarities, Rails 3 introduces the `Responder` to simplify the code further. For example, a few lines of code is all that's needed for a `create` action:

```
class PostsController
  respond_to :html, :xml

  def create
    @post = Post.create(params[:post])
    respond_with(@post)
  end
end
```

In this snippet, `respond_with(@post)` routes to `show` to display the new record if `@post` was saved successfully, or to `new` if the object failed validations, for example.

Again, this is just a small sampling. You can find examples of these new features and more in the next section, in which you build a Rails 3 application from scratch.

A first Rails 3 application

To run Rails 3, your system must have either Ruby version 1.8.7 or Ruby version

1.9.2 or a newer release of the programming language and its attendant libraries and interpreter. It is also beneficial to have the Git software version control system installed on your machine, as Rails 3 and many other influential Rails projects are maintained in Git. Your system should also have a database engine, such as SQLite (version 3), MySQL, or PostgreSQL. A Web server is not required to develop a Rails application, but it's usually part of a production deployment.

To create a Rails 3 application, you must have the Rails 3 prerelease gem and all its dependencies. At the moment, you can install the required components with just a few commands (see Listing 1). (Check the documentation for Rails 3 before you continue, as the specifics may change from release to release.)

Listing 1. The Rails 3 prerelease gem and dependencies

```
$ gem install rails3b
Due to a rubygems bug, you must uninstall all older
versions of bundler for 0.9 to work
Successfully installed mime-types-1.16
Successfully installed mail-2.1.2
Successfully installed text-hyphen-1.0.0
Successfully installed text-format-1.0.0
Successfully installed memcache-client-1.7.8
Successfully installed rack-1.1.0
Successfully installed rack-mount-0.4.7
Successfully installed abstract-1.0.0
Successfully installed erubis-2.6.5
Successfully installed i18n-0.3.3
Successfully installed tzinfo-0.3.16
Successfully installed bundler-0.9.5
Successfully installed thor-0.13.1
Successfully installed rails3b-3.0.1
14 gems installed

$ gem install arel --pre
Successfully installed activesupport-3.0.0.beta
Successfully installed arel-0.2.pre
2 gems installed

$ gem install rails --pre
Successfully installed activemodel-3.0.0.beta
Successfully installed actionpack-3.0.0.beta
Successfully installed activerecord-3.0.0.beta
Successfully installed activeresource-3.0.0.beta
Successfully installed actionmailer-3.0.0.beta
Successfully installed railties-3.0.0.beta
Successfully installed rails-3.0.0.beta
7 gems installed
```

The next step is to generate the application—a small wiki, shown in Listing 2. The application creates and administers articles. Each article has a title and some prose, and you create a new article simply by creating a reference to it from the body of an existing page. A reference is any camel case word, such as *TheSolarSystem* or *TheOscars.*

Note: The source code for the wiki application is available from the Download table below.

Listing 2. The wiki Rails application

```
$ rails wiki
```

If you run `ls -lR` to see the contents of the application, a few new files stand out:

- *Gemfile* is the gem manifest mentioned earlier. At a minimum, the file must contain two lines: one to point to the source of the Rails 3 beta gem and one to bundle the Rails 3 beta gem itself. You probably want a third line (at least) to connect to a database:

```
source 'http://gemcutter.org'
gem "rails", "3.0.0.beta"
gem "sqlite3-ruby", :require => "sqlite3"
```

- *config/application.rb* contains many of the options previously found in config/environment.rb. The latter remains but is largely deprecated. One significant addition to config/application.rb is the *generators block*:

```
config.generators do |g|
  g.orm             :active_record
  g.template_engine :erb
  g.test_framework  :test_unit, :fixture =>
true
end
```

  Your Rails 3 application can use one of a number of compatible object-relational mappers (ORM), template engines, and test frameworks. The generators block specifies your preferences for the application and invokes the proper generator for your models, views, and so on.

- *db/seeds.rb* is not new to Rails 3, but it's important to mention, because it was added fairly recently (it was introduced in Rails version 2.3.4). If your application requires initial data to run properly, such as an administrative user, price codes, or static pages, create that data in db/seeds.rb and run the task `rake db:seed`. Prior to the seed file, no convention existed for initialization, and many developers put code in migrations, clouding the differentiation between creating the database and populating it.
- *config.ru,* found in the root of each Rails 3 application, is a so-called *rackup* file, or a configuration for a Rack-based application. Rails 3 is a Rack application and is compatible with any Web server that also supports Rack. In general, you need not touch config.ru unless you want to add other Rack components.

There are a few other new files; most, though, should seem familiar from Rails version 2.3. The config/routes.rb file serves the same purpose as before, albeit with

a much-simplified and more Ruby-like flavor. You'll see an example momentarily.

After you generate the application and edit Gemfile to capture your dependencies, your next step is to collect the gems your application requires. That's the job of the new utility, `bundle` (see Listing 3).

Listing 3. Collect the required gems

```
$ bundle
installFetching source index from http://gemcutter.org
Resolving dependencies
Installing abstract (1.0.0) from system gems
Installing actionmailer (3.0.0.beta) from system gems
Installing actionpack (3.0.0.beta) from system gems
Installing activemodel (3.0.0.beta) from system gems
Installing activerecord (3.0.0.beta) from system gems
Installing activeresource (3.0.0.beta) from system gems
Installing activesupport (3.0.0.beta) from system gems
Installing arel (0.2.1) from rubygems repository at
http://gemcutter.org
Installing builder (2.1.2) from system gems
Installing bundler (0.9.7) from rubygems repository at
http://gemcutter.org
Installing erubis (2.6.5) from system gems
Installing i18n (0.3.3) from system gems
Installing mail (2.1.2) from system gems
Installing memcache-client (1.7.8) from system gems
Installing mime-types (1.16) from system gems
Installing rack (1.1.0) from system gems
Installing rack-mount (0.4.7) from system gems
Installing rack-test (0.5.3) from system gems
Installing rails (3.0.0.beta) from system gems
Installing railties (3.0.0.beta) from system gems
Installing rake (0.8.7) from system gems
Installing sqlite3-ruby (1.2.5) from rubygems repository
at
    http://gemcutter.org with native extensions
Installing text-format (1.0.0) from system gems
Installing text-hyphen (1.0.0) from system gems
Installing thor (0.13.3) from rubygems repository at
http://gemcutter.org
Installing tzinfo (0.3.16) from system gems
Your bundle is complete!
```

The `bundle` utility, short for *Bundler,* downloads and installs all the gems named in Gemfile and any of those gems' prerequisites (see Listing 4). The `bundle` utility can also copy all those dependencies into your application, making your code base self-sufficient. Specifically, if you run `bundle pack`, the Bundler copies the corpus of gems to vendor/cache.

Listing 4. Running the bundle utility

```
$ bundle pack
Copying .gem files into vendor/cache
  * bundler-0.9.7.gem
  * thor-0.13.3.gem
  * abstract-1.0.0.gem
  * mime-types-1.16.gem
```

```
  * text-hyphen-1.0.0.gem
  * rack-mount-0.4.7.gem
  * rake-0.8.7.gem
  * text-format-1.0.0.gem
  * tzinfo-0.3.16.gem
  * rack-test-0.5.3.gem
  * builder-2.1.2.gem
  * erubis-2.6.5.gem
  * memcache-client-1.7.8.gem
  * rack-1.1.0.gem
  * sqlite3-ruby-1.2.5.gem
  * i18n-0.3.3.gem
  * activesupport-3.0.0.beta.gem
  * arel-0.2.1.gem
  * mail-2.1.2.gem
  * activemodel-3.0.0.beta.gem
  * activerecord-3.0.0.beta.gem
  * actionpack-3.0.0.beta.gem
  * railties-3.0.0.beta.gem
  * actionmailer-3.0.0.beta.gem
  * activeresource-3.0.0.beta.gem
  * rails-3.0.0.beta.gem

$ ls vendor/cache
abstract-1.0.0.gem         memcache-client-1.7.8.gem
actionmailer-3.0.0.beta.gem mime-types-1.16.gem
actionpack-3.0.0.beta.gem rack-1.1.0.gem
activemodel-3.0.0.beta.gem rack-mount-0.4.7.gem
activerecord-3.0.0.beta.gem rack-test-0.5.3.gem
activeresource-3.0.0.beta.gem rails-3.0.0.beta.gem
activesupport-3.0.0.beta.gem railties-3.0.0.beta.gem
arel-0.2.1.gem               rake-0.8.7.gem
builder-2.1.2.gem         sqlite3-ruby-1.2.5.gem
bundler-0.9.7.gem         text-format-1.0.0.gem
erubis-2.6.5.gem         text-hyphen-1.0.0.gem
i18n-0.3.3.gem               thor-0.13.3.gem
mail-2.1.2.gem               tzinfo-0.3.16.gem
```

Think of vendor/cache as your application's own gem repository. You can move the code base anywhere and have the gem software and versions you depend on—no remote repositories required. For example, if you run `bundle install` after `bundle pack`, the gems are installed from your application repository to your system (see Listing 5).

Listing 5. Installing the gems

```
Fetching source index from http://gemcutter.org
Resolving dependencies
Installing abstract (1.0.0) from .gem files at
  /Users/strike/projects/rails3/wiki/vendor/cache
Installing actionmailer (3.0.0.beta) from .gem files at
  /Users/strike/projects/rails3/wiki/vendor/cache
Installing actionpack (3.0.0.beta) from .gem files at
  /Users/strike/projects/rails3/wiki/vendor/cache
...
Installing thor (0.13.3) from .gem files at
  /Users/strike/projects/rails3/wiki/vendor/cache
Installing tzinfo (0.3.16) from .gem files at
  /Users/strike/projects/rails3/wiki/vendor/cache
Your bundle is complete!
```

Working on the wiki

To create the application, generate a scaffold for a page, create the database, seed the database with an initial page, and set up the necessary routes (see Listing 6). To keep things simple, a wiki page record is limited to a handful of fields: a title, a slug (an abbreviation of the title), a body, and timestamps to record when the page was created and when it was last modified. The title and slug are string fields; prose is a text field; and the timestamps are date and time fields. (Of course, a real wiki would have additional fields, such as the most recent author and previous revisions of the page. For brevity, this example also omits users and sessions, formatting, and any kind of authentication and authorization.) You can generate an initial model, a set of views, and a controller with the command `rails generate scaffold`.

Listing 6. The full wiki application

```
$ rails generate scaffold page title:string slug:string
body:text --timestamps
      invoke  active_record
      create    db/migrate/20100221115613_create_pages.rb
      create    app/models/page.rb
      invoke    test_unit
      create      test/unit/page_test.rb
      create      test/fixtures/pages.yml
       route  resources :pages
      invoke  scaffold_controller
      create    app/controllers/pages_controller.rb
      invoke    erb
      create      app/views/pages
      create      app/views/pages/index.html.erb
      create      app/views/pages/edit.html.erb
      create      app/views/pages/show.html.erb
      create      app/views/pages/new.html.erb
      create      app/views/pages/_form.html.erb
      create      app/views/layouts/pages.html.erb
      invoke    test_unit
      create      test/functional/pages_controller_test.rb
      invoke    helper
      create      app/helpers/pages_helper.rb
      invoke      test_unit
      create        test/unit/helpers/pages_helper_test.rb
      invoke  stylesheets
      create    public/stylesheets/scaffold.css
```

If you're wondering what happened to `./script/generate`, recall that it's now subsumed by the omnipotent `rails` command.

Run `rake db:create db:migrate` to create the database:

```
$ rake db:create db:migrate
==  CreatePages: migrating
=============================================
-- create_table(:pages)
   -> 0.0010s
==  CreatePages: migrated (0.0011s)
```

```
==========================================
```

The wiki exists now, but it's empty. Add an initial page to serve as an anchor for all other pages. Edit the file db/seeds.rb, and write code to create a new page, as shown in Listing 7.

Listing 7. The wiki anchor page

```
Page.create(
  :title    => 'The Marx Brothers Wiki',
  :slug     => 'Home',
  :body     => 'An encyclopedic guide to the Marx
Brothers.')
```

Run `rake db:seed` to execute the code. You can verify the page with a quick glance using `rails console`, as shown in Listing 8.

Listing 8. Verify the anchor page

```
$ rake db:seed
(in /Users/strike/projects/rails3/wiki)

$ rails console
Loading development environment (Rails 3.0.0.beta)
irb(main):001:0> Page.all
=> [#<Page id: 1, title: "The Marx Brothers Wiki", slug:
"Home",
    body: "An encyclopedic guide to the Marx Brothers.",
    created_at: "2010-02-21 12:24:43", updated_at:
"2010-02-21 12:24:43">]
```

Before proceeding with the code, set up the routes. Two routes are required: a default route to find the home page and another route to find a page by its slug. Listing 9 shows the final config/routes.rb file.

Listing 9. config/routes.rb (final)

```
Wiki::Application.routes.draw do |map|
  resources :pages
  root :to => "pages#show"
end
```

The `rails generate scaffold page` line in Listing 6 automatically created the route in line 2, which is RESTful. You must add the route in line 3 manually. The syntax to specify a default "root" of the site route is new in Rails 3. Line 3 says, "Map the route '/' to the 'show' method of the pages controller." The code for the `show` method finds the home page in the database and displays it.

After adding the new root route, delete the file public/index.html to preclude conflicts:

```
$ rm public/index.html
```

Now, turn your attention to the page controller. The code for a controller in Rails 3 can be exceedingly spartan. Listing 10 shows the initial implementation of the controller, with a sole `show` method.

Listing 10. A Rails 3 controller

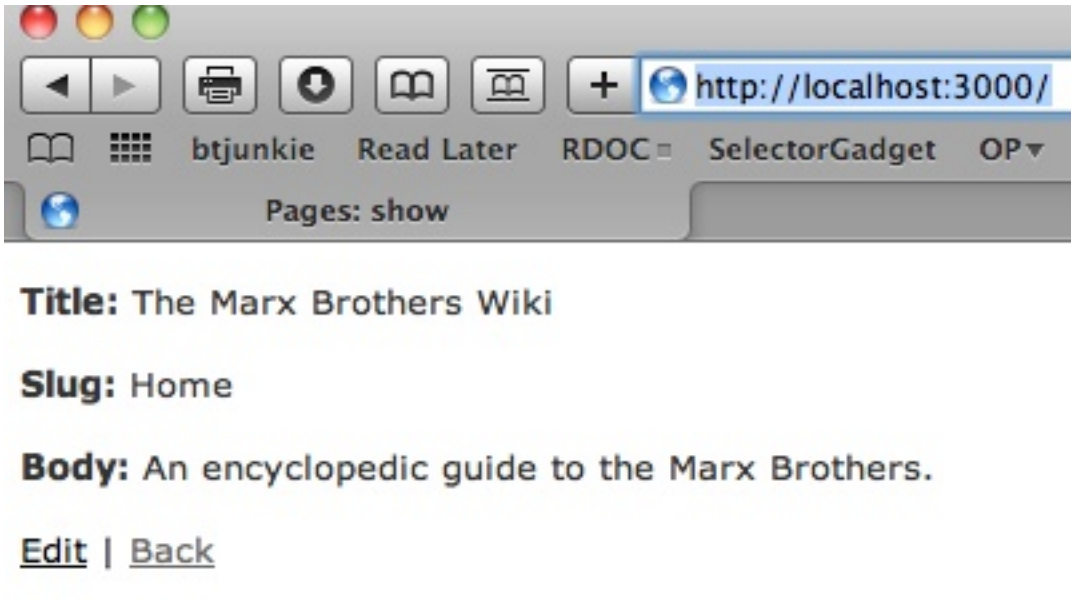
```
class PagesController < ApplicationController
  respond_to :html

  def show
    @page = Page.where( :slug => ( params[:id] || 'Home' )
).first
    respond_with( @page )
  end
end
```

As you can see, all the boilerplate typically found in a Rails 2 controller is missing. `respond_to` lists the formats the controller supports; here, it responds solely to requests for HTML. `respond_with` is shorthand for the logic to decide how the controller should proceed.

The syntax for the query is also quite different. The lookup is an example of the Rails 3 relational algebra. You may be wondering why the `first` suffix is required. `where` and other operands that express the query do not actually cause the query to execute. Instead, the query sits idly by until the data is actually needed. This is *lazy loading,* or deferring the query as long as possible. `first` sparks an actual inquiry from the database.

If you run the application now, you should see something similar to Figure 1.

Figure 1. The Rails 3 wiki application



Now, add more code to the controller. Listing 11 shows the complete controller.

Listing 11. The complete Rails 3 controller

```
class PagesController < ApplicationController
  respond_to :html
  before_filter :get_page, :except => [ :create ]

  def create
    respond_with( @page = Page.create( params[ :page ] ) )
  end

  def edit
  end

  def index
    render :action => :show
  end

  def show
    @page ||= Page.new( :slug => params[ :id ] )

    if @page.new_record?
      render :action => :new
    else
      respond_with( @page )
    end
  end

  def update
    @page.update_attributes( params[ :page ] )
    respond_with( @page )
  end

  private

    def get_page
      @page = Page.where( :slug => ( params[:id] || 'Home'
) ).first ||
        Page.where( :id => params[:id] ).first
```

```
    end
end
```

In the controller, the `index` method merely reflects the `show` action with no page identifier, thus rendering the home page. `show` displays a page, given an ID or a slug (the lookups for all actions are centralized in `get_page`, further reducing the amount of code); if a page does not exist, a new page is prepared for editing.

The `Page` model merely validates that all its fields are present:

```
class Page > ActiveRecord::Base
  validates_presence_of :body, :slug, :title
end
```

The work to translate the camel case references to links to other pages occurs in the view for the `Page` model. A helper function in app/helpers/pages_helper.rb does the heavy lifting, keeping the view minimal (see Listing 12).

Listing 12. The camel case translation helper function

```
module PagesHelper
  def wikify( page )
    return '' if page.body.blank?
    page.body.gsub(
/^([A-Z][[:alnum:]]*([A-Z][[:alnum:]]*)+)/ ) do |match|
      link_to( $1, :action => :show, :id => $1 )
    end
  end
end
```

The view is typical, as shown in Listing 13.

Listing 13. A typical view

```
<p>
  <b>Title:</b>
  <%= @page.title %>
</p>

<p>
  <b>Body:</b>
  <%= raw wikify( @page ) %>
</p>

<%= link_to 'Edit', edit_page_path(@page) %> |
<%= link_to 'Back', pages_path %>
```

The `raw` operator is new to Rails 3. Counter to previous releases of Rails, all strings are emitted safe, stripped of HTML, by default. If you want to emit a string with HTML, you must use `raw`.

Switching Rails

Beyond the improvements and conveniences shown here, Rails 3 offers better performance than its predecessors, especially in rendering partials. You can also create your own validator classes and take advantage of more streamlined standard validations. For instance, this validation, written by Jeremy McAnally, once required four separate lines of code:

```
validates :login, :presence => true, :length => {:minimum
=> 4},
  :uniqueness => true, :format => { :with =>
/[A-Za-z0-9]+/ }
```

The Rails Guides, the official tutorials for Rails, are currently being updated for Rails 3. You can also find extensive instruction and clever solutions on the blogs of Jeremy McAnally, Yehuda Katz, Gregg Pollack, and other community leaders (see Resources). A number of popular books are under revision for the new release, too, including the seminal "pickaxe" book, *Agile Web Development with Rails* (see Resources).

Downloads

Description	Name	Size	Download method
Source files for the example wiki	wiki.zip	120KB	HTTP

Information about download methods

Resources

Learn

- Rails 3 release notes: Read the release notes for a comprehensive and up-to-date list of additions, enhancements, and changes in the code.
- Merb: Rails 3 integrates many of the features of the Merb framework. Discover Merb's history and features.
- Jeremy McAnally's blog: Read this blog for tutorials and expert insights.
- Yehuda Katz's blog: Katz is a core Rails contributor and a chief architect of the integration of Merb and Rails.
- Gregg Pollack's blog: Pollack hosts a weekly podcast and occasional screencasts about Rails.
- *Agile Web Development with Rails* (Sam Ruby, Dave Thomas, David Heinemeier Hansson, et al., The Pragmatic Bookshelf, 2009): This book is an invaluable resource if you want to learn Rails.
- developerWorks Web development zone: The Web development zone is packed with tools and information for Web 2.0 development.
- IBM technical events and webcasts: Stay current with developerWorks' technical events and webcasts.

Get products and technologies

- Ruby on Rails: Visit the Rails home page to read more about Rails and to download the software.
- Sunspot: Download a copy of the Sunspot text search plug-in for Rails from Github.
- Clearance: This prepackaged Rails solution helps with user authentication.
- Bullet: Incorporate Bullet into your application to find queries that are too lazy or too eager.
- Oink: Try the aptly-named Oink to reduce the memory usage of your Rails application.
- IBM product evaluation versions: Download these versions today and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- developerWorks blogs: Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Martin Streicher
Martin Streicher is a freelance Ruby on Rails developer and the former Editor-in-Chief of *Linux Magazine.* Martin holds a Masters of Science degree in computer science from Purdue University and has programmed UNIX-like systems since 1986. He collects art and toys. You can reach Martin at martin.streicher@gmail.com.