Presentation to

eBIG
Web Development SIG

Learning Ruby on Rails

Here there be Magick!

(with apologies to Piers Anthony)

July 23, 2007

Why are you here?

- Check the hype?
- Learn how to get started?
- Learn some new tricks/
- …???

What’s your background?

- Never looked at Rails?
- Played around with it?
- Developed a public site?
- Guru?
- …?

My background

- Started programming in 1967 – UF Computer Center:
    - My own water-cooled (personal) IBM 360 computer!
- Many different languages/systems: Assembler language, APL, C, CICS, Cobol, Control RDBMS, DBASE, Fortran, MS Access, Paradox, Perl, PL/1, REXX, Theos, Unix, Visual Basic…
- Mostly management consulting for the past 15+ years

What I wanted to do

- Build database-backed web applications
- On my list since before Web 2.0
- Tried GoLive/Dreamweaver <ugh!>
- Looked at some commercial systems <$!>
- Tried Perl frameworks <ugh!>
- Tried writing Ajax direct <whew!> (simple app)
- Tried Rails <mmm! (sorta)>

My approach

- Read a bunch for starters
  - *Build Your Own Ruby on Rails Web Applications* <good starting point>
  - *Agile Web Development with Rails* <heavy going in places but this is the one book you gotta have>
  - *Ruby for Rails* <good 2nd book>
  - *Ajax on Rails* <too specialized for beginners>
  - *Programming Ruby* *<the classic 'PickAxe' book>*
  - *Rails Recipes* <great ideas… some actually work!>
  - *Rails Cookbook* *<basic, starter samples…good>*
  - *Ruby Cookbook* *<solid code samples>*
  - *Pragmatic Version Control Using Subversion*
- Follow along with book's code
  - Only good up to a certain point
  - Lots of code that doesn't work
- Build a usable application and make it public!
  - 4mypasswords.com

Our webapp criteria

- Must be
  - Something I would use
  - Offered to the public
  - Free of ongoing administration
  - Memorable domain name (short…)
  - Relative quick/easy to implement in a basic form
  - Have an option for a revenue model if successful

4MyPasswords.com

- Place to store sensitive information on the web
  - (e.g., passwords, credit card info, registration or serial number information, …)
- SECURE
- Easy to use

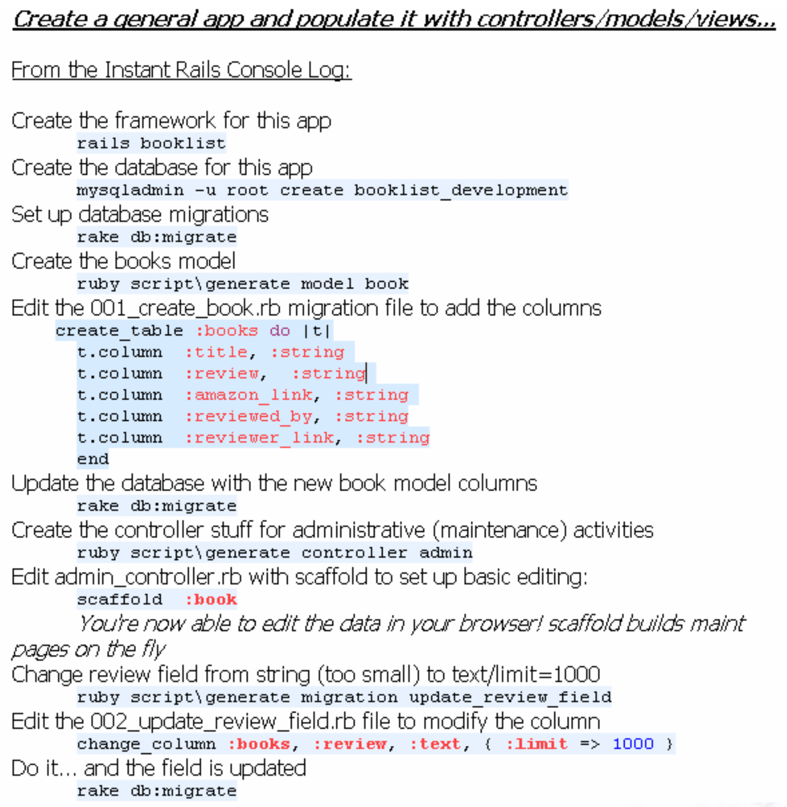
- <u>For me</u>: replace Coda Hale's *Genius* utility that I've been using for years
  - Too many different machines I use
  - At different client locations

1st Step…

- Pretty straight-forward:
  - Get the tools (Instant Rails, Aptana/RadRails, Firebug, Source Chart, Session Manager, Web Developer)
    - (too bad: I like Vim)
  - Understand the MVC approach (tutorial)
  - Create an application structure
  - Create a simple database
  - Create a simple interface (scaffolding!)
  - Actually use it for stuff!
    - Internal "Ideas" application
    - Internal "Booklist" application

1st Results…

- Create a simple app in about 1 hour
- Cool!



Create a general app and populate it with controllers/models/views…

From the Instant Rails Console Log:

Create the framework for this app
```
    rails booklist
```
Create the database for this app
```
    mysqladmin -u root create booklist_development
```
Set up database migrations
```
    rake db:migrate
```
Create the books model
```
    ruby script\generate model book
```
Edit the 001_create_book.rb migration file to add the columns
```
  create_table :books do |t|
    t.column  :title,  :string
    t.column  :review,   :string
    t.column  :amazon_link, :string
    t.column  :reviewed_by, :string
    t.column  :reviewer_link, :string
    end
```
Update the database with the new book model columns
```
    rake db:migrate
```
Create the controller stuff for administrative (maintenance) activities
```
    ruby script\generate controller admin
```
Edit admin_controller.rb with scaffold to set up basic editing:
```
    scaffold  :book
```
You're now able to edit the data in your browser! scaffold builds maint pages on the fly

Change review field from string (too small) to text/limit=1000
```
    ruby script\generate migration update_review_field
```
Edit the 002_update_review_field.rb file to modify the column
```
    change_column :books, :review, :text, { :limit => 1000 }
```
Do it… and the field is updated
```
    rake db:migrate
```

Easy to scratch the surface…

STEEP learning curve after that!

(As easy as falling off a cliff)

“Any sufficiently advanced technology is indistinguishable from magic.”

Arthur C. Clarke

2nd Step…

- Going beyond the 1st simple steps was much harder/more time consuming…
  - Spent ***TONS*** of time Googling for info…
  - Collected many cheat sheets from the web…
  - Started creating my own cheat sheets for specific tasks…
  - Rails wiki deluged with porn spam (wiki.rubyonrails.org)
    - *NB – This has since been fixed with Wiki security*
  - Forums: Some great help; some non-responsive; many assumptions about how much people know
  - East Bay Ruby Meetup Group – Lots of help!
    - Bottom line: a great community
  - RailsPlayground.com – EXCELLENT support while I was getting things up and running (and I needed it :-) !

Taking that 2nd step…

- Get as much ‘off the shelf’ as possible:
  - User/Login system:
    - LoginGenerator (replaced by Acts_as_authenticated)
  - Improved scaffolding:
    - Ajaxscaffold.com (now ActiveScaffold.com)
  - Encryption/decryption:
    - EzCrypto.rubyforge.org
  - Rails webhosting:
    - RailsPlayground.com
  - CSS template:
    - minimalistic-designs.com
- SSL – Thawte certificate

It works – and folks can use it!

https://www.4mypasswords.com/

nubyrubyrailstales.blogspot.com/

What I ~~like~~ love about Rails…

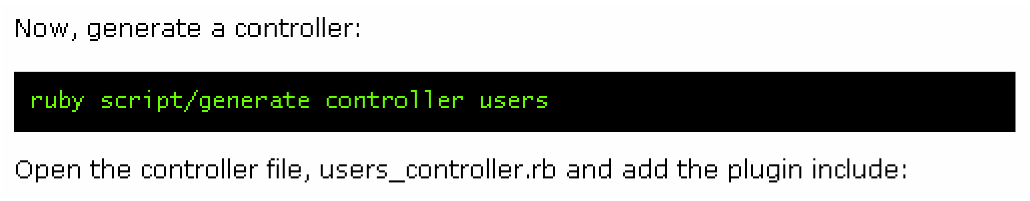
- Framework & scripts
- Migrations
- WEBrick / Mongrel
- Conventions…?
- ~~Pluralization…?~~
- { Wiki / Api }.rubyonrails.org
- Documentation; e.g., api.rubyonrails.org
- ActionMailer
  - But it's a little weird

What I ~~like~~ love about Rails…

- Deprecation warnings…
  - Rails is a very young framework
  - There is intense development going on to extend the framework
  - There are *many* changes happening, not necessary all documented nor blindingly obvious…
- Pretty good tools (e.g., Aptana, …)
  - But I wish there were a Vim option
  - VI Improved (www.vim.org) has a *great* Rails plugin, but no visual directory tree presentation

…and what I wonder about

- What happened to script/help?
- ~~Pluralization…?~~
    - Conventions not followed:



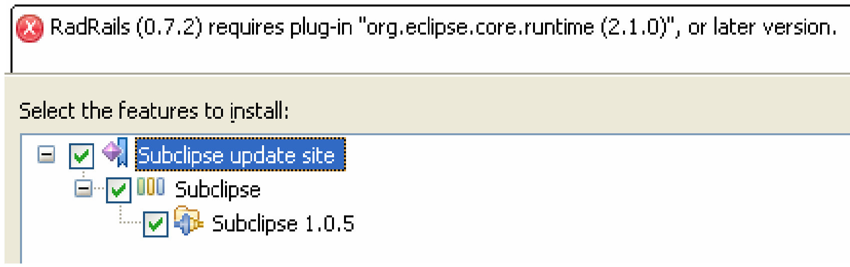
Now, generate a controller:

```
ruby script/generate controller users
```

Open the controller file, users_controller.rb and add the plugin include:

- …and CamelCase versus camel_case
    - script/generate ProductBacklog ≠> product_backlog
- …and just who is it "intuitive" to… Non-English speakers?
    - belongs_to *:user*
    - has_many *:users*
    - has_and_belongs_to_many *:users*
- …and then there are collections
    - *:client* versus *:clients* ?

… and what I wonder about (2)


RadRails (0.7.2) requires plug-in "org.eclipse.core.runtime (2.1.0)", or later version.

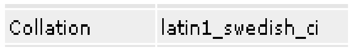
Select the features to install:

- ✓ Subclipse update site
  - ✓ Subclipse
    - ✓ Subclipse 1.0.5

- Things are broken:
  - RadRails update fails
  - Ruby update fails
  - *NB: Now fixed with Aptana!*
- Installing a plugin is counter-intuitive:
  - Sites say "do this" (script/plugin install…)
  - RadRails says "do this" (plugin create…)
  - *Finally*: just copy the stuff into the right directory!
- Fast changing
  - Deprecations – I'm just learning; why do I have all these recommended solutions with warnings…?

… and what I wonder about (3)

- Gaining familiarity with all the "auto-magical" stuff
- Soooo many special helpers, RJS, methods, …
- Database collation defaults (not Rails?)

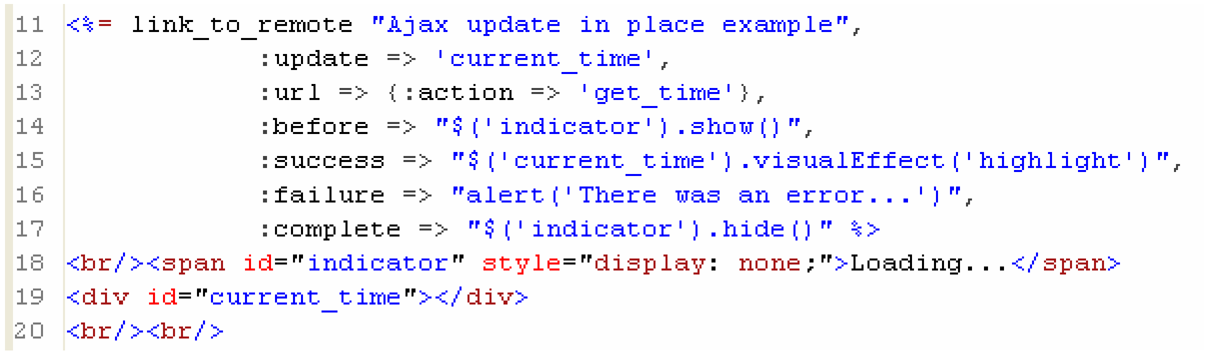


Collation	latin1_swedish_ci

- Documentation
  - Scattered (rubyonrails.org, rubyforge.org, individual websites, …)
  - Inconsistent
  - Missing
- Search engine visibility…?
- MySQL implementation: pure Ruby vs C gem…?
- Mysterious interactions… (e.g., Rails Recipe 34)

… and what I wonder about (4)

- **Claim that you don’t need to know Javascript…**
  - **Well, you certainly have to be able to handle “snippets” of Javascript, as well as understand Protocol and script.aculo.us commands!**



```
<%= link_to_remote "Ajax update in place example",
            :update => 'current_time',
            :url => {:action => 'get_time'},
            :before => "$('indicator').show()",
            :success => "$('current_time').visualEffect('highlight')",
            :failure => "alert('There was an error...')",
            :complete => "$('indicator').hide()" %>
<br/><span id="indicator" style="display: none;">Loading...</span>
<div id="current_time"></div>
<br/><br/>
```

To be done…

- Subversion – DONE!
- Testing – Just getting started
- ActiveScaffold…?
- CSS/GUI interface updates
- User interface/functionality
    - E.g., password changes…
- Capistrano – Got it working!
- Documentation
- XML Export/Import
- …?

Reasons I prefer Rails over .NET

(Jeff, on 'Softies on Rails')

1. **Ruby**. The language is just awesome for object-oriented development. Remember, I was a longtime C++ developer, and C# after that, so I've always loved statically-typed OO languages. But I usually get more done with Ruby in less time.
2. **ActiveRecord**. The easiest ORM I've ever used (again, it's mainly because of Ruby's language features that make this possible)
3. **Forced MVC design**. There are other great architectures for the web, but for database-backed apps, MVC is fine 80% of the time; so for that sweetspot, Rails makes it easy.
4. **TDD support**. To call it "support" is to understate it. Rails expects TDD, and so it's a first-class citizen in the application skeleton. The best Rails developers I know all use TDD. It's the only framework I know that doesn't just "allow" you to do TDD, it assumes you ARE doing TDD, and makes it easy to do so.
5. **Ajax support out of the box**. And in a clean way that again leverages Ruby to its fullest. There's almost no mental context switch between writing Ajax and non-Ajax code - it's all Ruby, same idioms, same "feel" of where your code should go.
6. **Agile development baked in**. Like TDD, everything about a Rails app skeleton screams for best practices, and it goes out of its way to induce you to keep your code DRY, refactor often (this is why the TDD aspect is so important), and build incrementally.
7. **Limited choices coerce you into following Rails' best practices**. Some people call it the "opinionated" side of Rails. I call it standing on the shoulders of giants who've already figured out good ways to stitch together the various tiers of a web app. The REST support in Rails is a great example of how average developers become good developers if they follow Rails opinion on how you should think about your application.
8. **Database agnosticism**. There's built-in support for, I don't know, about eight popular databases, and it's almost 100% transparent.
9. **OS agnosticism**. I develop Rails sites on Windows as easily as I do an a Mac or Linux (ok, I sort of take that back; the tool support isn't quite there on Windows, and the refusal of Microsoft to include a gnu-compatible C compliler with Windows keeps guys like me behind the rest of the pack).
10. **It's fun**. Sounds weird, I know. But it's not just me saying that. The ease with which I can start building an app and see results, with tests from the start, make it more fun to work on Rails apps.
11. **It's all free**.
12. **The Rails core is kept to a minimum**. There's more power with a lot less API "surface area" than any other framework I know. Most Rails developers don't need intellisense, because it's much easier to just know what to do; and when you're not sure, everything is so consistent between classes, it's much easier to just guess the right thing to do.
13. **Plugins from the community**. Awesome.

What’s the Verdict?

- Rails is awesome…
- …and has some maturing to do

- And I love it!

“Rails is full of magic, and database connectivity is a particularly magical area of the framework.”

Rails Recipes

Recipe #15, p65

Chad Fowler

Building a Simple Application

July 23, 2007

ActiveRecord magic…

- User.<col_name> for all columns defined
- User.id
- Basic CRUD methods (new, create, save, update, update_attributes, delete, destroy, …)
- User.find(…) - :first, :all, :conditions => …
- User.find_by_<col_name>
- User.find_by_<col_name1>_and_<col_name2>
- User.find_all_by_<col_name>
- Average, maximum, minimum, sum, count
- …

More ActiveRecord magic…

- When you create relationships between models, you get lots more methods. Assume you have the following:

User Class	has_many :accounts
Account Class	belongs_to :users

- Then you get:
  - @account.user.<col_name> for every column in the users table
- You can also create a many-to-many…

Article	has_many :readings has_many :users, :through => :readings
User	has_many :readings has_many :articles, :through => :readings

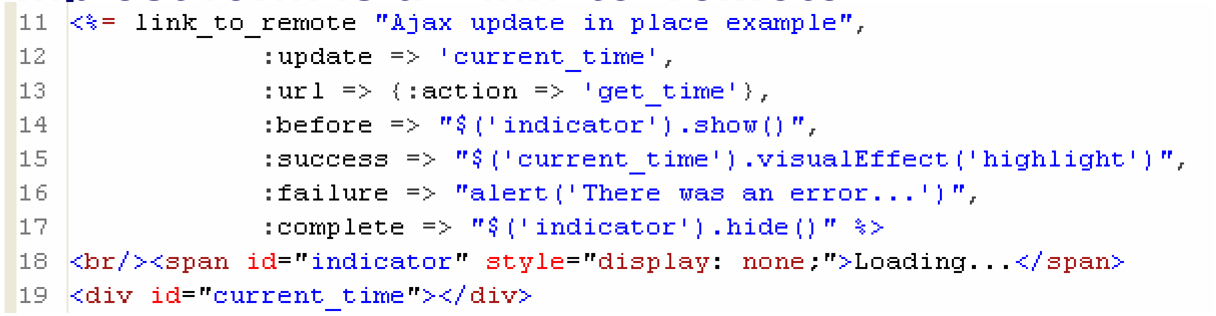
ActiveRecord Migrations…

- Create a migration either through creating a model or explicitly
- Update your database/table definitions as you need to
- Back out changes that didn't work
- Modify data during a migration if you need to upgrade the data

- *One of my favorite features in Rails*

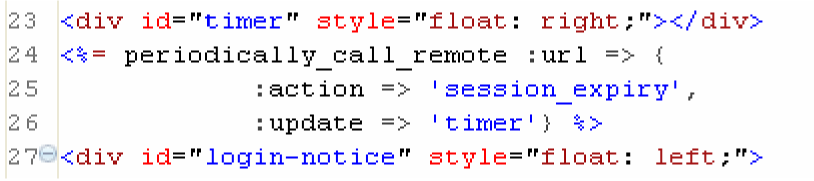
Doing some Ajax…

- Simplest form is a “link to remote”



```
<%= link_to_remote "Ajax update in place example",
            :update => 'current_time',
            :url => {:action => 'get_time'},
            :before => "$('indicator').show()",
            :success => "$('current_time').visualEffect('highlight')",
            :failure => "alert('There was an error...')",
            :complete => "$('indicator').hide()" %>
<br/><span id="indicator" style="display: none;">Loading...</span>
<div id="current_time"></div>
```

- Then there’s the “periodically_call_remote”

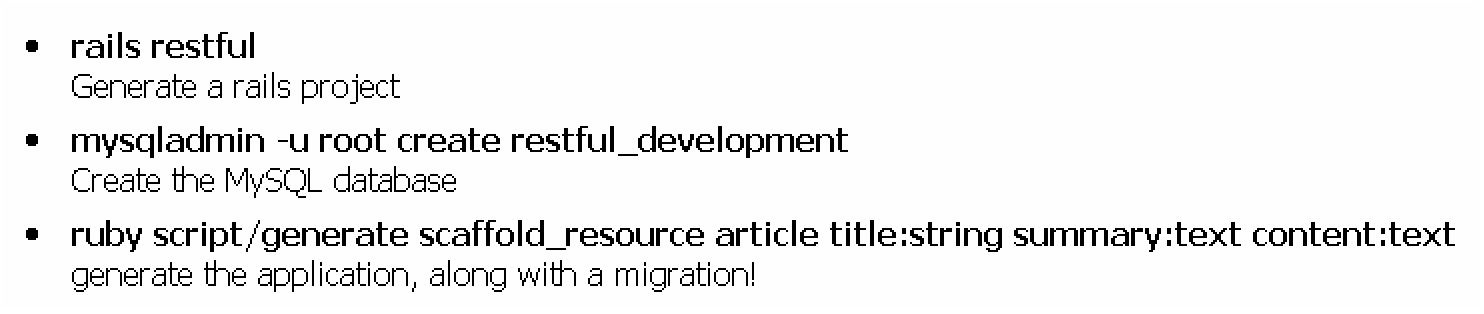


```
<div id="timer" style="float: right;"></div>
<%= periodically_call_remote :url => {
            :action => 'session_expiry',
            :update => 'timer'} %>
<div id="login-notice" style="float: left;">
```

- And… “link_to_function”, “remote_function”, “observe_field”, “observe_form”, “form_remote_tag”, “remote_form_for”, “render :partial =>”, …

RESTful development…

- **Simple to setup the basic structure:**



- rails restful
  Generate a rails project
- mysqladmin -u root create restful_development
  Create the MySQL database
- ruby script/generate scaffold_resource article title:string summary:text content:text
  generate the application, along with a migration!

Trade for services?

- I've got a SuSE Linux machine running, and would like to get it set up for development.
- I've got a running Sun Classic X system for trade.

Jon Seidel, CMC®

Jseidel AT edpci DOT net