

Rails Magazine

fine articles on Ruby & Rails

Interview with **YEHUDA KATZ**
by Rupak Ganguly

Rails Background Processing
By James Harrison

Generating PDF with ODF templates
by Rodrigo Rosenfeld Rosas

Ruby
Kaigi
2009

RubyKaigi Exclusive Coverage

RubyKaigi 2009 Roundup
by Ehab El-Badry

Interview with **MATZ**
by Muhammad Ali

Interview with **KOICHI SASADA**
by Muhammad Ali

Interview with
DAVID HEINEMEIER HANSSON
by Mark Coates

Feel the Radiance with
Radiant CMS
by Saurabh Bhatia

Interview with
THOMAS ENEBO
by Rupak Ganguly

Oracle Tips and Tricks
by Greg Donald

THE FUTURE OF RAILS

Don't listen to us, listen to them.

RPM gives your developers **transparency** into the performance details of our applications, **saving significant time** in isolating bottlenecks.

— Ian McFarland, Pivotal Labs

New Relic's RPM has **dramatically improved** the way we monitor the health and performance of our applications.

— Mark Imbriaco, 37signals

If you are running a Rails app of any reasonable size **you have to use New Relic.**

There is no way around it.

— Tobias Lütke, Shopify

With RPM we had **more time to add new features** and fix issues, rather than scanning through piles of log files.

— Rick Olson, Lighthouse

We used the other tools out there, and were **not happy until we switched to New Relic.** I was amazed how much we found in just a few hours of monitoring. We got the **real-time picture we had been dreaming about.**

— Yaroslav Lazor, railsware



New Relic RPM lets you see and understand performance metrics in real time so you can fix problems fast. It's intuitive. It's granular. And, it's a 10-second Rails plug-in install.

Take **RPM Lite** for a spin — it's free!

NewRelic.com

A Word from the Editor

by Olimpiu Metiu

Welcome to this new installment of Rails Magazine! While not the most technical nor the largest, we believe this still is one of the best ones so far.

Thanks to all authors and editors who made this issue possible! Carlo Pecchia recently joined the editorial team, we are thankful to have him on board. If you like the magazine, we invite you to consider contributing an article or joining as an editor.

While there are a variety of techniques for PDF generation, automation with OpenOffice is a novel approach discovered by Rodrigo Rosas, which works well for fairly complex layouts or when supporting non-technical users. We'll cover additional techniques for PDF generation in the next issues.

Background processing becomes more important for Rails developers, and James Harrison presents a solution using `delayed_job`. Our next issue will present an alternative technique.

Olimpiu Metiu is a Toronto-based architect and web strategist.

He led the Emergent Technologies group at *Bell Canada* for the past couple of years, and his work includes many of Canada's largest web sites and intranet portals.

Olimpiu is currently a senior architect at *Research in Motion* (the maker of *BlackBerry*),

where he is responsible with the overall architecture of an amazing collaboration platform.

A long-time Rails enthusiast, he founded Rails Magazine as a way to give back to this amazing community.



Follow on **Twitter**: <http://twitter.com/olimpiu>

Connect on LinkedIn: <http://www.linkedin.com/in/olimpiu>

Saurabh Bhatia introduces the Radiant CMS, a first article in a mini-series on Radiant programming.

We asked DHH, Yehuda and Thomas Enebo to share their thoughts on Rails 3 and beyond. DHH goes a step further and hints at what to expect beyond it.

Finally, special thanks to Muhammad Ali and Ehab El-Badry for their coverage of Ruby Kaigi 2009. I hope you will enjoy reading their interviews with Matz and Koichi as much as I did.

DISCUSS: <http://railsmagazine.com/4/1>

Contents

A Word from the Editor	3
by Olimpiu Metiu	
Background Processing with <code>Delayed_Job</code>	4
by James Harrison	
Generating PDF with ODF templates.....	8
by Rodrigo Rosenfeld Rosas	
Interview with Yehuda Katz	13
by Rupak Ganguly	
Interview with David Heinemeier Hansson	16
by Mark Coates	
Feel the Radiance with Radiant CMS	18
by Saurabh Bhatia	
Interview with Thomas Enebo.....	21
by Rupak Ganguly	
Oracle Tips and Tricks.....	23
by Greg Donald	
Ruby Kaigi – Exclusive Coverage	24
RubyKaigi 2009 Roundup	25
by Ehab El-Badry	
Interview with Matz	28
by Muhammad Ali	
Interview with Koichi Sasada.....	30
by Muhammad Ali	



Background Processing with Delayed_Job

by James Harrison

Introduction to background processing

If you've written an application before, chances are you ended up wanting to do something while the user waited for that thing, that could potentially take a while. It might be generating a PDF, sending out bulk emails, grabbing information from an API, or something that involves a lot of data which is slow to load. In these cases, you can take advantage of background processing; chopping this operation into a job or task, giving it to a background worker that processes it outside of the web request, and returning the data to the user later through AJAX or simply in the data your site shows. Let's take a real-world example to explore this a bit better.



James Harrison is currently reading Computer Science at Royal Holloway, University of London and develops webapps in his spare time using Rails. His websites include EVE Metrics (<http://www.eve-metrics.com>), a market analytics tool for the virtual world EVE Online working with millions

of transactions, with hundreds of thousands being added every day. In any time left over, he does live sound engineering and helps out with the technical side of things at his local theatre.

I have an application which takes a user's API key and user ID and gives it to the game EVE Online. This application then makes around 10 calls to the API with these credentials, stores around 3,000 rows of data, and the report generated is then made available to another user of the site who has requested this report. With this API-fetching stage potentially taking tens of seconds, or more if the API servers are being slow or are down, it makes sense to pull the data-retrieving stage out to a separate task which is triggered by the user. The user enters their details, they are checked with a single quick poll to ensure the credentials are good (with a short timeout in case the API servers are down), and are either given an error or told that their report is being generated. Once started, the job is stored in the database with their credentials, and the job runs, eventually spitting out a fresh report into the database which shows up in the user's interface.

Notably not all of the task at hand is done by the background job, a quick check is done to inform the initial feedback to the user. Background jobs are typically only used where the task will take several seconds; long enough to annoy the user or tie up application servers.

Delayed_job and other plugins

There are several choices for plugins to use in background processing. BackgroundDRb is one of the most mature plugins but doesn't work at all on Windows, making developing with it difficult if you run Windows on your development environment. It does however support crontab-style automated tasks, saving you tinkering with the cron tool directly to run automatic jobs on a daily or hourly basis.

Nanite is another library which is somewhat more Rails-independent and thus a little more complex to get working than others, so it's not a great choice for beginners. If you need a huge amount of flexibility, you can set up RabbitMQ (the Erlang job queue it requires), and can run your code in an EventMachine-supporting environment (which at the time of writing rules out Passenger-hosted sites), then you can have a look at this fantastically fast and flexible tool. Both BackgroundDRb and Nanite are great tools and worth considering when choosing a worker, but because they are more complex to get started with, this guide will focus on delayed_job. Many of the concepts in delayed_job are replicated in other plugins, so hopefully you can still use some of the advice and information in this guide when working with these tools.

Delayed_job is a very flexible, small plugin which has several advantages: it's pure Ruby, it uses an ActiveRecord job queue, and is easy to hack on/modify for your own needs if it doesn't support what you need out of the box. These features make it ideal for most cases and great for beginners. It's distributed as a Rails plugin, making installation a cinch, though at the time of writing it requires you to add a migration to your database manually.

The plugin is split into several sections: the Delayed::Job class which represents a job, and the Delayed::Worker class which is responsible for getting jobs and working through them. We'll take a close look at the concepts in delayed_job and other background processing tools, and then look under the hood to see how DJ implements those concepts.

Delayed_job Concepts

The underlying concepts of delayed_job are simple enough to understand and should appear familiar to those who have used other queue-based background workers. This is one of the plugin's key strengths; it is a very simple plugin that is easily extended and adapted.

Delayed::Job is the class that represents a single job. It subclasses ActiveRecord::Base, and is backed onto the table

delayed_jobs. This table contains locked_at/_by columns which are used for workers to lock jobs to work on them, a run_at and failed_at pair which define a job's status, and created_at/updated_at columns. As well as this, there is a field which stores the serialized struct; this lets delayed_job know what class to call perform on.

Other classes are fairly token in understanding the concepts, other than Delayed::Worker. This class wraps a simple loop which performs Job#work_off, which we'll look at in more detail next.

Under the hood and failure conditions

The Job#work_off method is fairly simple, it gets a given number of jobs (It's only parameter, by default 100), and works through the stack. It reserves each job, calls the perform method on the job, and counts successes/failures. Some more of the logic behind the scenes is found in Job#reserve, which handles actually locking the job in question. It accepts a block, and provides the job it has reserved as a variable to that block. Essentially, this handles locking rather neatly without having to mess about with the messier parts of that particular problem if you want to write or adapt your own worker methods to perform specific jobs and so on.

Failure conditions are to be expected in development and aren't something to overlook in production; delayed_job handles this with a last_error column which stores the traceback from any error that occurs in job processing. However, if a job fails, it is retried. If that job is going to send 10,000 emails, you probably don't want it to repeat if it gets halfway through and hits a dodgy email, you'd rather just have it send out half and complain at you quietly, rather than sending 5,000 emails out 20 times to the same people. It's not a recommended path to happy customers, even if you ignore the server load issue!

Fortunately, Job has a constant called: MAX_ATTEMPTS. Change this and you can have it try once before giving up. It also has a class attribute: destroy_failed_jobs. By default this is true, meaning job queues don't end up with broken jobs. However, it can make debugging tricky, and I'd recommend changing this in development. You may want to use something like the following instead of the default of true:

```
self.destroy_failed_jobs = (ENV['RAILS_ENV'] == 'production' ? true : false)
```

This will set the variable accordingly based on your environment settings. If you use a plugin such as exception_notifier or a tool like Exceptional, you won't get notified about errors in your worker methods as delayed_job rescues these and stops an exception being raised. You can handle exceptions manually by adding your code (such as sending an email) in the rescue section of Job#work_off's reserve begin/rescue block. Out of the box, delayed_job does not support sending emails on errors or similar error reporting.

An example worker

Let's look at an example worker. Let's say we want to receive a YAML-serialized object from another site using a web hook:

```
class ProcessPushData < Struct.new(:raw_object)
  def perform
    object = YAML::load(raw_object)
    # Do stuff with object
  end
end
```

That's as simple as they get! It's really that easy, all we need is a class that responds to a perform method, and we're good! From within this class you have access to your Rails environment, such as other models. So if we now wanted to create some new models from that deserialized object, it's not a problem.

Working with Delayed::Job

Jobs are all well and good, but how do we add new ones? Here's a simple example from an application I'm using delayed_job on:

```
Delayed::Job.enqueue MarketOrderUploadJob.
new(params[:log], @user.id, @key.id)
```

We simply call the enqueue method, passing it a new instance of the class, initialized with the variables it expects. enqueue then serializes this and stores it to the queue to be picked up by a worker. We can get a little more complex with something like this:

```
Delayed::Job.enqueue(1, Time.now+1.day){ MarketOrderU-
ploadJob.new(params[:log],
  @user.id, @key.id) }
```



This shows two features: priorities and delayed jobs. The former feature is self explanatory; the higher the priority, the faster that job will be picked up by a worker (the job finds SQL orders by priority in descending order). The delayed job feature allows you to set jobs which will run in the future, which can be useful in some cases but is typically not needed.

Of course, typically we'll want to have some sort of interface for viewing our job queue's status. Easy!

```
@jobs = Delayed::Job.find(:all, :order => 'id DESC')
```

Delayed::Job is just another ActiveRecord descended class, so we can treat it as a model to some extent. There's a few tricks you might find useful when dealing with jobs in your views, though.

```
j.deserialize(j.handler).class #=> The name of the class this worker is using
```

```
j.last_error #=> The last error this job raised during processing
```

You can also show a job's status if locked_at is set, the job is being worked on. If last_error is set then the job hit an error and could be flagged for investigation.

There's another common pattern you may wish to use when working with data a job should return. If you are generating a report via a delayed job, you can poll that job or the report using AJAX on the page, displaying a loading indicator while the report is generated while ensuring a snappy response for the user. The easiest way to implement this is to add a Boolean flag to your report's model, but you can also record the job ID and look at the job status that way.

Testing and deployment strategies

Testing your delayed_jobs is easy enough, simply run the perform method in your tests and see if it does what you'd expect. It's best to keep the testing of delayed_job itself separated from your workers, of course, so add them into your test suite separately from the plugin's RSpec tests.

Keeping the workers organized in your source can be a huge help. Personally I like to keep mine in app/jobs, and use:

```
config.load_paths += %W( #{RAILS_ROOT}/app/jobs )
```

to get Rails to load the classes within the folder into the environment so delayed_job can use them.

Deployment of the workers can be more complex. On server-side all you need to do is to call a rake task, jobs:work, to work off jobs. I use God, a Ruby-based process monitor and manager, to manage my workers. God's typical recipes can be used for this, and a full recipe can be found in the resources section at the end of this article. The only slightly tricky bit is the start command:

```
w.start = "rake -f #{RAILS_APPLICATION_ROOT}/Rakefile jobs:work RAILS_ENV=production"
```

You can of course run multiple workers, which will name themselves by hostname and process ID by default. There's one other neat trick you can do with delayed_job; let's say you have a very time-critical background job that needs doing as soon as possible, as well as a bunch of not-so-important slow-running jobs. You can priorities, which will help to some extent, but even better is to have a worker dedicated to these jobs. Pick a priority, and specify MIN_PRIORITY on the command line when you start up the worker:

```
w.start = "rake -f #{RAILS_APPLICATION_ROOT}/Rakefile jobs:work RAILS_ENV=production MIN_PRIORITY=2"
```

This will make this worker ignore all jobs except those with a priority of 2 or above. You can also use the MAX_PRIORITY variable for even tighter control. Using these simple flags you can build complex worker setups that are easily managed from within God or another service manager of your choice.

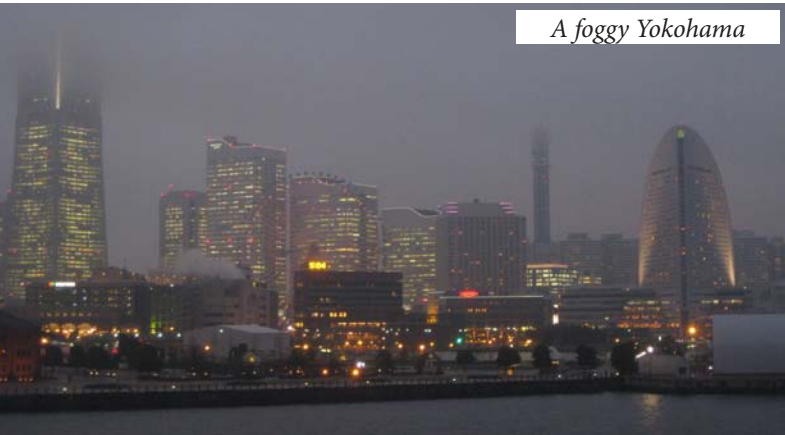
Now you've got your job written, tested, workers set up on your server, all that remains is to deploy, restart your workers, and enjoy the extra flexibility of a background processing queue.

Resources

- delayed_job - http://github.com/tobi/delayed_job
- nanite - <http://github.com/ezmobius/nanite>
- BackgroundDRb - <http://github.com/gnufied/backgrounddrb>
- God: <http://god.rubyforge.org/>

CODE: http://www.talkunafraid.co.uk/railsmag-delayed_job

DISCUSS: <http://railsmagazine.com/4/2>



A foggy Yokohama



Discover

the latest in the technological realm +

Interact

With our latest contributions, as we continuously find ways to get developers out of tight spots +

AdVance

your coding with simple tweaks and ideas that enhance your overall performance

Throw on a white coat and take a look in the minds of our team of developers, where there is always a new solution being concocted.

@space

is a rising champion in the Ruby on Rails development arena. In our labs, new ideas are brewing, solutions are being uncovered, and everything is out in the open.

Generating PDF with ODF templates

by Rodrigo Rosenfeld Rosas

There are several ways of generating PDF output with Rails. Specifically with Ruby or better yet, on all frameworks and languages. Some techniques include templates written in LaTeX[1] or DocBook[2], while others require manual generation of the entire PDF. That, actually, is not a pragmatic solution for most real-world environments, nor does it follow the MVC separation principle.

Background and Motivation

At the beginning of 2007, I started working for FAPES (<http://www.fapes.es.gov.br>), a government foundation that incentives research, science and technology in Espírito Santo, the state where I live. I was given the job to create the site for our freshly created foundation. It was created altogether with SECT, the Secretary of Technology in our state, around 2005, without even a physical place available.



Rodrigo Rosenfeld Rosas is an electrical engineer, living in Vitória-ES, Brazil. He developed a C++ framework for Real-Time mobile robotic systems based on Linux+Xenomai for his master thesis in 2006, at UFES. For validating purposes, he also developed a robot and a real-time framgrabber module. He currently works at Geocontrol (<http://www.geocontrol.com.br>) and has found Ruby in 2007, while evaluating

Rails for web developing, after previous experience with C/C++, Delphi, Perl, among other languages.

He loves his wife and has no children yet (kids are great, but no hurry). Enjoys surfing, brazilian music (samba, choro, seresta) and playing guitar, and he is trying to learn other instruments: cavaco, bandolim, pandeiro, flute and violin (maybe sax and clarinet in the future).

E-mail: rr_rosas@yahoo.com.br.

Since it is a small foundation, I was responsible for everything related to IT, including network infrastructure, helpdesk and programming. At the time, I was asked to provide maintenance to NOSSABOLSA, a web system implemented in ASP, which was (and still is) an important program of FAPES/SECT for financing studies in private colleges for students of low income families. I knew nothing about ASP or Rails at the time.

My previous serious web programming experience (several years before this job) was with Perl. I didn't work with web programming for lots of years after that. At the middle of 2007, I had to develop a new site for promoting an event

we were organizing that needed a subscription system, along with lots of static information. Since I had a Linux server available, I didn't have to implement it in ASP, which would take a lot of days and I only had a week for developing the new site.

I started to then look at the web for available modern web frameworks, that allowed me to develop the website faster. I took a look at Perl frameworks, of course, J2EE, .Net, TurboGears, Django, Rails and a lot more. It became clear that Rails was the right choice. So I learned Rails, and developed the whole site in a week. Since it would be a temporary site, I didn't mind trying a new framework at the time. I was so excited by Rails and specially Ruby, that, when we needed a new permanent site for FAPES, it was the logical choice.

Shortly after the site creation, there was a need to generate some contracts based on input from web forms that could be printed and delivered to FAPES. There were lots of problems generating the contracts in HTML. CSS was not well thought for printing. It was difficult (if at all possible) to setup headers & footers and actually the print depends a lot on browser configurations and rendering engines. As a result, the printed versions would not follow a unique layout, which was a problem to us. So, I started thinking about PDF output. I read all the usual PDF generation techniques but none of them seemed to fit my needs.

Proposed Solution Overview

The problem was that there were a lot of types of contracts and they were a bit long, which would take a lot of time for preparing them without a good template system. I knew the juridic department was unable to give me LaTeX or DocBook-formatted contract models. They only knew MS Word, and I had to live with it. There was also not a lot of time for implementing contracts generation.

It seems that when under pressure, we are extremely creative. Fortunately, I remembered that ODF was, actually, a XML file, along other files in a folder structure that were zipped in an ODF file. I extracted the file and took a look at a special file, called content.xml. Then I realized that it was possible (and pretty easy) to replace some special text templates with form fields submitted to the web server. It was also pretty easy to import MS Word documents in OpenOffice.org[3] and save them in ODT format. And I didn't have to teach the juridic department any new document writing technique, such as Latex or DocBook. They could just use what they were used to.



Additionally, there were some free tools that could convert odt to pdf, using OpenOffice.org in a “headless” environment, which meant that I could run it as a background daemon without even having a graphical environment installed. This daemon could be run in the same server as the application or in another dedicated server, if you prefer. Here is a possible usage for setting OpenOffice.org as a daemon, listening on port 3003:

```
soffice -accept="socket,host=localhost,port=3003;urp"
-norestore -headless -invisible -nofirststartwizard&
```

Then, any UNO[4] enabled software could convert any ODF file to any OpenOffice.org support output format, including PDF. For instance, one could use PyODConverter[5] for converting an odt document to pdf:

```
/opt/openoffice.org3/program/python DocumentConverter.
py document.odt output.pdf
```

Just be sure to edit DocumentConverter.py, changing the port to reflect the one OpenOffice.org is listening at, since the port can't be currently passed as a parameter. Following the above example, it means:

```
DEFAULT_OPENOFFICE_PORT = 3003
```

The overall idea is illustrated on the following pictures:



New student

Name:
 Address:

items	
Item: Books	Value: 90.49
Item: Airline Tickets for International Congress	Value: 4000
Item: <input type="text"/>	Value: <input type="text"/>
Item: <input type="text"/>	Value: <input type="text"/>
Item: <input type="text"/>	Value: <input type="text"/>

ODF template + HTML form



Approved Items	
Item	Value (US\$)
Books	90.49
Airline Tickets for International Congress	4000.0
Total	4090.49

Name of Director
Account Information

Final generated output

A Possible Implementation

I was pretty happy for having found the solution to my problems and only needed some little time to implement the solution. This technique, actually, can be very easily implemented in any language. At least in Ruby, all seem easy to implement. Here is what I currently use for generating PDF contracts (save it to config/initializers/contract.rb):

```
require 'rexml/document'
module Contract
  CONST_FIELDS = {'DirectorName' => 'Name of Director', 'FAPES_Account' => 'Account Information'}
  CONTRACTS_URL=' /contracts'; CONTRACTS_DIR = Rails.
  public_path+CONTRACTS_URL
  ATTACHMENTS_DIR = "#{CONTRACTS_DIR}/attachments/"
  OUTPUT_DIR = "#{CONTRACTS_DIR}/generated/"; TEMPLATES_DIR =
  "#{CONTRACTS_DIR}/templates/"
  class << self
    # Save the template file at public/contracts/templates/scholarship.odt, then call:
    # Contract::generate('scholarship', {'student_name' => 'Rodrigo Rosenfeld'}, 'scholarships/1')
    # public/contracts/generated/scholarships/1.pdf will be created. Output is PDF link address.
    def generate (template_file, fields, output_file, options={})
      fields.merge!(CONST_FIELDS)
```

```

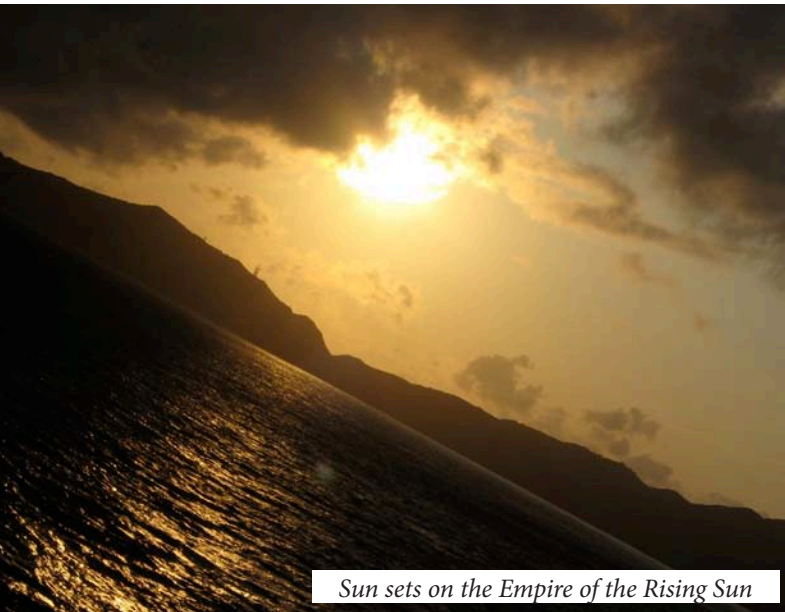
template_file = TEMPLATES_DIR+template_file+'.
odt'
# replaces non-alphanumerics to underscore. Security is responsibility from calling method.
output_file.gsub! /[\^\\\w\.-]/, '_'
pdf'
pdf_output_filename = OUTPUT_DIR+output_file+'.
pdf_output_filename_temp = options[:attachment]
? OUTPUT_DIR+output_file+'_without_attachment.pdf' : pdf_output_filename
output_file = OUTPUT_DIR+output_file+'.odt'
output_dir = File.dirname(output_file); FileUtils.mkdir_p(output_dir)
Kernel.system "unzip -o #{template_file} content.xml -d #{output_dir}" or return nil
content_file = "#{File.readlines("#{output_dir}/content.xml")}"
# before replacing expressions, generate table templates
options[:tables].each {|t| content_file=generate_table(content_file,
t[:table_name], t[:line], t[:fields])} if options[:tables]
replace_expressions(content_file, fields)
File.open("#{output_dir}/content.xml", 'w') {|f| f.write content_file }
FileUtils.cp_r template_file, output_file
#pdf-converter is a script, that currently uses PyODConverter (DEFAULT_OPENOFFICE_PORT=3003):
#/opt/openoffice.org3/program/python /usr/local/bin/DocumentConverter.py $@
Kernel.system("zip -j #{output_file} #{output_dir}/content.xml; pdf-converter #{output_file} #{pdf_output_filename_temp}") or return nil
return merge_pdf(pdf_output_filename_temp, ATTACHMENTS_DIR + options[:attachment], pdf_output_filename)

```

```

if options[:attachment]
pdf_output_filename.sub /\Apublic/, ''
end
def replace_expressions(str, fields)
# The pattern "#{student_name#U}" will be replaced by 'RODRIGO ROSENFELD'
str.gsub!(/#\{(.*)#\}/) do
result = (fields[$1] or '')
case $3
when 'U'; result.mb_chars.upcase.to_s
when 'd'; result.mb_chars.downcase.to_s
when 'C'; result.mb_chars.capitalize.to_s
# lots of other formatters here for writing number at full length, as currency, etc.
else; result # doesn't change
end
end
end
# this generates dynamic tables into ODF Templates. It is necessary to define a name for the table in OpenOffice.
# 'line' starts at 1.
# generate_table(content_string, 'Items', 2, [{'n' => 1, 'item' => 'Desktop computer'}, {'n' => 2, 'item' => 'Laser printer'}])
def generate_table(content_xml, table_name, line, fields)
document = REXML::Document.new(content_xml)
template_line = document.root.elements["//table:table[@table:name='#{table_name}']/table:table-row[#{line}"]].to_s
document.to_s.sub(template_line, fields.collect

```



Sun sets on the Empire of the Rising Sun



Bonsai banzai!


```
{|f| replace_expressions(template_line.dup, f)}.join)
  end

  # returns url to merged file or nil if it couldn't
  be generated
  def merge_pdf(contract, attachment, output)
    [contract, attachment].each {|f| return nil un-
less File.exist?(f)}
    Kernel.system("pdftk #{contract} #{attachment}
cat output #{output}") or return nil
    output.sub(Rails.public_path, '')
  end
end
end
end
```

The implementation is not really important and I will not talk very much about it, since I am pretty sure a lot of plugins with better options and implementation will be developed using this technique. Most of the implementation is trivial to understand. Optimizations can be made, of course. For instance, it should not be necessary to unzip content.xml from the template before each conversion.

I'll just take some time to explain some parts that might not be obvious to all readers.

```
template_line = document.root.elements[“//table:table[@
table:name='#{table_name}']/table:table-row[#{line}]”].
to_s
```

```
document.to_s.sub(template_line, fields.collect {|f|
replace_expressions(template_line.dup, f)}.join)
```

It is intended to support simple table templates. Patterns should be written in one line, which will be replaced for several lines containing a collection of data taken from the web, such as Items to Purchase, or whatever. It is necessary to name the table (see table properties in OpenOffice) and tell the method which line should be used as template. First line is number one.

There is also a helper method for merging PDFs, so that one special PDF can be attached at the end of the dynamic one if it is necessary. The implementation could be changed to accept other parameters for specifying a header and a footer PDF. The implementation uses pdftk[6] for merging them.

As you probably noted, you would write ODT files with patterns such as “I, #{student_name#U}”, agree...”, that will be replaced by “I, RODRIGO ROSENFELD, agree...”, when “Rodrigo Rosenfeld” is submitted to the web server, in a form.

Test Cases

You should also write an unit test for assuring the output is correctly generated. Here is a possible unit test (test/unit/contract_test.rb):

```
require 'test_helper'

class ContractTest < ActiveSupport::TestCase
  # This test is not definitive, since it doesn't test
  the generated pdf content. But chances are good to be cor-
  rect if content.xml is correct and
  # the generated file is a PDF (starts with '%PDF').
  However this doesn't test if the merge is good, although
  it assures a lot of steps are working.
  test 'generate contract' do
    FileUtils::rm_rf Rails.public_path + '/contracts/
generated/scholarships/test/'
    assert_equal '/contracts/generated/scholarships/
test/rodrigo.pdf',
      Contract::generate('scholarship', {'student_
name' => 'Rodrigo Rosenfeld', 'address' => 'R. Nascimento
Silva, 107',
      'total' => 4090.49}, 'scholarships/test/rodrigo',
      :tables => [[:table_name => 'Items', :line => 2,
      :fields => [[:name => 'Books', 'value'=> 90.49],
      {'name' => 'Airline Ticketings for International
Congress', 'value' => 4000.00}]]],
      :attachment => 'scholarships/contract_terms.pdf')
```



Feeding the soul



Tradition meets timeless sea


```
assert FileUtils::identical?(Rails.root.to_s + '/
test/fixtures/expected_content.xml',
  Rails.public_path + '/contracts/generated/schol-
arships/test/content.xml')
f = File.new(Rails.public_path + '/contracts/gen-
erated/scholarships/test/rodrigo.pdf')
assert_equal '%PDF', f.read(4) # Is the output a
pdf?
end
end
```

Conclusion

Generating PDF from ODF templates proved to be pretty easy. Enjoy your free time, now that you can save a lot of it manually preparing PDF generation! Or use it for writing a good plug-in for PDF generation using this technique. :)

Recently, a new plugin for ODF generation, written in Ruby, is available at <http://github.com/sandrods/odf-report/tree/master>. This plugin uses the idea presented in this article for ODF generation, using the rubyzip gem for zipping/unzipping, instead of launching an external program for this task. For readers interested in implementing a plugin for PDF generation in Ruby, I would recommend taking a look at this ODF generator and adapt it to integrate with OpenOffice.org and PyODConverter as demonstrated in this article.

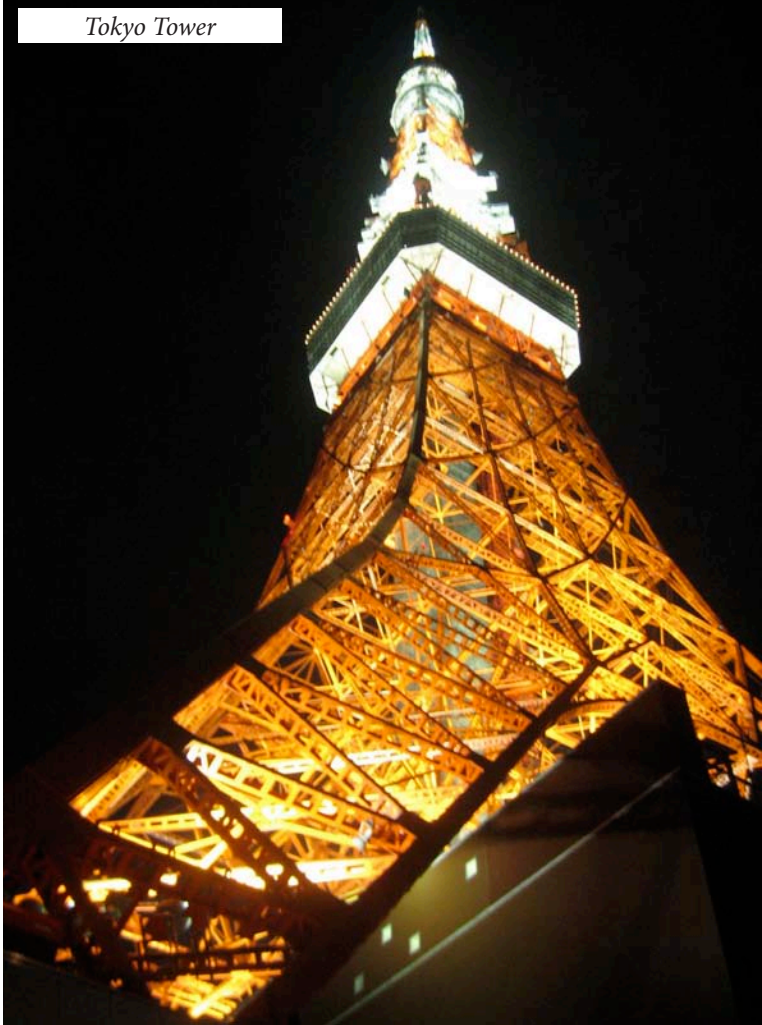
Resources

- [1] Latex - <http://www.latex-project.org>
- [2] DocBook.org - <http://www.docbook.org>
- [3] OpenOffice.org - <http://www.openoffice.org>
- [4] UNO - <http://wiki.services.openoffice.org/wiki/Uno>
- [5] PyODConverter - <http://www.artofsolving.com/opensource/pyodconverter>
- [6] pdftk - <http://www.pdftk.com/pdftk>

CODE: http://github.com/railsmagazine/rmag_downloads

DISCUSS: <http://railsmagazine.com/4/3>

Tokyo Tower





Interview with Yehuda Katz

by Rupak Ganguly on August 15th, 2009

Rupak: Will most of the appealing features of Merb be merged into Rails? Will the Merb framework be dropped after that? If not, what is the goal of the Merb platform now and in future?

Yehuda: Pretty much everything that we knew and loved about Merb is slated to be a part of Rails at some point in the future or made available as a plugin. However, there are still a number of applications using Merb, and we will continue to support it as long as there is interest.

Rupak: With all the Rails + Merb development going on, what would your recommendation be for current Merb users: Move onto Rails or upgrade to Merb 1.1?

Yehuda: Once we release Merb 1.1, you're going to want to be on it. Hold off on upgrading to Rails until the transition story is complete and stable.

Rupak: What are other features/concepts that will be shipped with Rails 3?

Yehuda: There are a few categories of changes:

* **Improved internals.** Rails 3's internals have gone through a major working over in an effort to make them easier to understand and extend. As much as possible, parts of the Rails internals have had their object boundaries solidified and well-defined. This means that if you want to understand how ActionController interacts with ActionView, there is a well-defined boundary. If you want to replace one of the components or instrument the boundary, it is now very easy. This will help with Rails maintenance into the future, and will also make it a lot easier for plugin developers to write plugins that work well with future versions of Rails.

* **Performance.** Rails 3 has significantly lower overhead compared with Rails 2. For instance, the overhead of rendering partials or collections is between 20 and 25% of the overhead of doing the same operation in Rails 2.3. One specific example: rendering a collection of 100 partials is down from 8 milliseconds to just 800 microseconds. This is possible because we can leverage the crisper boundaries between components to cache common costs. For instance, we do a much better job of caching the template lookup for a given path, format (html), and locale (en).

* **New Paradigms.** Probably the most significant new paradigm is `respond_with`, imagined last January by DHH, and written by José Valim. This brings conventionality to RESTful controllers, wrapping up the default logic in a Responder object. Instead of a bunch of logic to decide how to

render an object (JS, JSON, or HTML – render or redirect) you simply say `respond_with(@comment)`. Like I said, this will use the Rails conventions by default, but it's possible to create your own Responder for a given controller or group of controllers that wraps up a different pattern. You can see a lot more about this in José's blog post announcing the feature (<http://blog.plataformatec.com.br/2009/08/embracing-rest-with-mind-body-and-soul/>)

* **Agnosticism.** Finally, Rails 3 will make it a core principle to decouple its implementation from the ORM, JavaScript library, caching backend, template engine, testing framework and more. We still want to have a fantastic getting started experience, with defaults that can bring you from 0 to 60 in no time at all. We also want it to be possible for users with different needs to use the tools they need to use to get the job done.

Rupak: Has the Rails Core team been looking at a possible release date for Rails 3?

Yehuda: We don't have a release date yet. A lot depends on how quickly we can get plugins running on Rails 3 and get through an RC period with most applications transitioning smoothly.

Rupak: Will there be any backward compatibility issues with Rails 2.x projects upgrading to Rails 3? Any recommendations?

Yehuda: Backwards compatibility has been a priority. However, it has been common for people to hack into Rails' internals to get common things done. Since we've done so much work on the internals, most of those hacks will no lon-

Session on Rails 3
by Yehuda Katz



ger work. The good news is that we're committed to providing public, supported APIs for people to use to do the same things.

There will also be some small changes. For instance, we have finally removed .rhtml, and clearly defined the semantics for template lookup in ActionView. This means that some things that may have worked accidentally before won't work now. On the bright side, we have now clearly defined the intended behavior, so changes that you make should carry you forward into the future..

Rupak: What are your initial thoughts about accelerating the adoption of Rails 3 when it is released?

Yehuda: I'd like to see plugins embrace Rails 3 so we can hit the ground running with a fully working suite of Rails plugins once we release.

Rupak: To change the subject a bit, How was Ruby Kaigi 2009? What are your impressions about the content presented and the people you met?

Yehuda: It was really great. I loved being able to meet some of the people I've heard about when doing Ruby work but never had the pleasure of meeting. The content was at a very high level – much higher than the average American Ruby conference. People understood Ruby well and weren't afraid to push its boundaries. The talks by the maintainers of various branches of Ruby were all very interesting as well.



Yehuda Katz is currently employed by Engine Yard, and works full time as a Core Team Member on the Rails and Merb projects. He is the co-author of jQuery in Action and the upcoming Merb in Action, and is a contributor to Ruby in Practice.

He spends most of his time hacking on Rails and Merb, but also on other Ruby community projects, like Rubinius and Datamapper. And when the solution doesn't yet exist, he'll try his hand at creating one – as such, he's also created projects like Thor and DO.rb.

Rupak: How did your presentation go? What was it about?

Yehuda: My presentation was about how parts of Rails 3 can be used in other contexts. I showed how ActiveRecord allows you to get validations on a standard Ruby object, how you build a controller from the ground up, and how you can use ActiveSupport safely as an extended Ruby standard library.



Rupak: Anything that you would like to say to the organizers of Ruby Kaigi 2009??

Yehuda: Great job! I'd love to see you in the United States at some of our conferences.

Rupak: What do you think are the main reasons for such slow adoption of Ruby 1.9? What features would you like to emphasize that would encourage it's adoption?

Yehuda: It has mostly been the lack of a stable transition path. Database drivers haven't cleanly worked, various popular libraries have had problems with encodings, and there are missing tools (like ruby-debug) that people rely on.

However, Ruby 1.9 is significantly faster than Ruby 1.8, is more memory efficient, and is generally built on more modern technology than Ruby 1.8. The problems that have caused slow adoption are, for the most part, resolved. Database drivers work, the popular libraries are encoding-aware, Rails 2.3 and above work on Ruby 1.9, and ruby-debug is finally ported.

Rupak: Any recommendations for people trying to be compatible with both Ruby 1.8 and 1.9?

Yehuda: The biggest current issue is constant scoping. You cannot assume that a constant available outside a block will be available inside it. This is because Ruby 1.9 changes the constant scope when a block is evaluated in a new context, unlike Ruby 1.8. For instance, when using RSpec, you cannot refer to constants that you created inside the "describe" block inside an "it" block.

Additionally, String#[] now returns a single-character String; in order to get the Integer value, you'll need to call .ord on the result. ActiveSupport has extensions to make forward-compatibility easier. For instance, we define String#ord on String in Ruby 1.8, so you can replace "Hello"[0] with "Hello".ord and get an Integer in both Ruby 1.8 and 1.9.

Rupak: What do you still miss in Ruby?

Yehuda: Interesting question. Over the past few weeks I've been hurting for better caching primitives (the general purpose Hash isn't perfectly suited for caching) but I was able

Interview with David Heinemeier Hansson

by Mark Coates on August 18th, 2009

Mark: Okay, we're dying for an inside scoop — can you give us any news about Rails 3? For instance, will Rails views support HTML 5 and CSS 3.x? Or anything new you can share?

David: Rails really doesn't say much about how you write your HTML or CSS. All we do is wrap a bunch of common HTML in helpers. You're always free to add your own and write whatever HTML and CSS you please. But we are indeed targeting HTML5 for the scaffolding templates and looking to make nice helpers around things like the new video and audio tags.

It's an exciting time for web developers. The momentum behind HTML5 and CSS3.x is building every day. We'll make sure that we're right there with the best support that we can offer.

Mark: Can you give us any insight on where you see not only Rails, but web development in general, going in the next five years?

David: I try not to engage in too much rubbing of the crystal ball. Especially when it comes to making long-term predictions. But considering that the basic building blocks of the web five years ago were HTML, CSS, and JS, I'll use yesterday's weather and predict that they'll continue to be the dominating technologies for implementing web applications. I don't have a lot of faith in vendor-driven alternatives like Silverlight or Flash.

In fact, I think these vendor technologies will have even less relevance as the new revisions of HTML and CSS and the speed increases we've seen for JS continues to expand what's possible to do with these tools. I don't think most people will bother with Flash for video, for example, in a few years. HTML should obviously be able to handle that on its own.

Mark: Is competition in the web development space ultimately good for Rails? What do you think is in competition to Rails and why?

David: Competition that leads to innovation is always a great thing. Rails made a lot of people wake up to the fact that most web development environments were terrible. It's great to see that many of these platforms have been making progress on digesting those ideals.

I think the main competition for Rails is the status quo. People who have no interest in changing what they've been doing for years. Getting web developers to move on from antiquated technologies should be the mission for all the

next-generation frameworks and libraries currently eyeing competition amongst themselves. We have much more to gain from increasing the size of the pie than from haggling over the crumbs that are up for grabs.

Mark: What do you think about Microsoft and ASP.net jumping on the MVC bandwagon and do you think Rails' success prompted their adoption?

David: Microsoft is late to the party as usual, but they can afford to be. Or at least they used to be able to afford it. Lots of places are so locked in to Microsoft technologies that they don't even look at anything else. It's good that these people are now finally being exposed to some of the ideas the open source world have been enjoying for a long time.

I absolutely think that Microsoft was acting reactionary. Not just to Rails but to many of the other modern platforms that are intent on improving life for developers.

Mark: What do you see as the second-best web development framework?

David: I don't use anything but Rails for actual development, so I'm not properly equipped to answer that. But I like what both Django and Seaside are doing.

Mark: Can you share your thoughts on getting Rails into 'enterprise' environments, specifically large, old guard organizations that are intimately tied to Microsoft? Moreover, is Rails ready for the enterprise?

David: I think this question is a few years stale. Rails is already deep inside tons of enterprise environments. I've found that the most important parameter for this penetration is just time. If you are big, old, stodgy organization, you're simply not capable (or interested) in being at the cutting edge. So anything that doesn't have a year long history behind it rarely gets considered. Thankfully, we're finally getting there with Rails, so it's getting easier for people every day.

Mark: Are you currently writing any books? Or planning to in the near future?

David: I actually just finished the manuscript on a new book called Rework that'll be published in spring next year. I wrote along with Jason Fried and Matt Linderman from 37signals. It's a more general, updated, awesome version of our previous book on starting a web business called Getting Real. Mighty excited about that!

Mark: Where is 37signals headed? Any new products on the horizon? Or any new technologies/frameworks brewing or being extracted from your projects?

David: We just announced that we've been working on a big integration project for quite a while called 37signals Accounts. Single sign-on, integration, suites, and all that good jazz. We also have a variety of internal exploration projects that we're not ready to talk about yet. But I'm sure we'll be able to extract some good stuff from that.

Real-time updates is one of the things we've been playing with that's pretty exciting.



David Heinemeier Hansson was born in 1979 in Copenhagen, Denmark. After he graduated from Copenhagen Business School in 2005, he moved to Chicago. He is a partner at 37signals (<http://37signals.com>), the company behind Basecamp, Highrise, Backpack, Campfire,

Writeboard, and Ta-da List.

37signals also runs a popular weblog called Signal vs. Noise. David is the creator of Ruby on Rails, which he has been working on since 2003. He won the Google and O'Reilly award for Best Hacker of the Year in 2005 for his work on rails and the Jolt award for product excellence for Rails 1.0. He also co-authored the successful books *Getting Real* and *Agile Web Development with Rails*, which have sold over 140,000 copies together world-wide.

Mark: Are you involved with anything outside of 37signals and Rails—any side commercial or open source projects, maybe?

David: I have my hands plenty full running 37signals, being involved with Rails, and carrying a functional social life on the side.

Mark: And finally, our readers are dying to know — pirates or ninjas?

David: Pirates, obviously.

DISCUSS: <http://railsmagazine.com/4/5>



Feel the Radiance with Radiant CMS

by Saurabh Bhatia

Amongst all the revolution happening in the Web 2.0 era, web publishing has been a revolution in itself. In recent times, the concept of web publishing has evolved itself into a major field of software development and have got eyeballs from all over the world. Content Management Systems have been the result of this movement both commercial and Open Source. Blogs, Wikis and CMS like Drupal, Joomla has become extremely popular for making it substantially easy to publish content over the web, and help the users focus on content rather than worrying about the code.

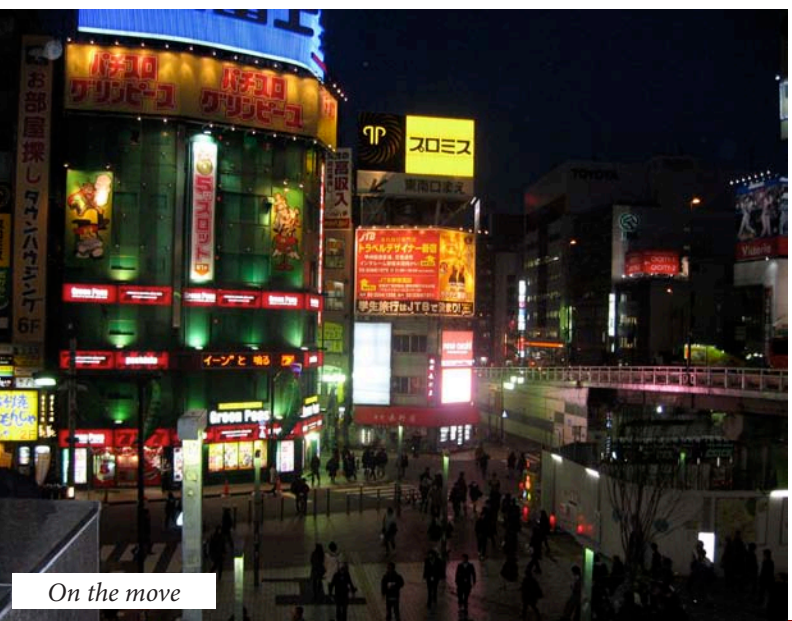


Saurabh Bhatia is Co-Founder and CEO of Safew Technology Labs. He has been working with Ruby and Rails since 2005. He loves to write software in Rails apart from managing his company.

He also has a knack for system administration and is an expert level Linux system

administrator. In his free time, he likes to listen to music and read fiction. He can be reached at saurabh@noisybrain.net.

Radiant CMS, as their website defines is “Radiant is a no-fluff, open source content management system designed for small teams”. This means it does not boast of too many amazing features out of the box but provides a light and flexible framework to develop with. As an introductory article, I will walk you through the installation of Radiant CMS and then familiarize you with the terminology associated with the system.



On the move

Installing Radiant

Anyone with even basic knowledge of Rails can go through the setup of Radiant in a breeze. I will start the installation on my local machine, which is an Ubuntu 8.10. The pre-requisites for installing Radiant on your machine are Ruby 1.8.6 or later, Rails 2.1, MySQL/PostgreSQL/SQLite3 and RubyGems.

```
saurabh@laptop:~$ gem install radiant
```

Once, the gem is installed, create your project directory and change directory to it:

```
saurabh@laptop:~$ mkdir radiant
saurabh@laptop:~$ cd radiant/
```

The radiant command with the database switch creates a Rails Radiant application for you. You can then select the database as MySQL, Postgres and SQLite3, which changes the config/database.yml file according to the selected database. In my case, my choice was MySQL.

```
saurabh@laptop:~/radiant$ radiant --database mysql /
home/saurabh/radiant/
```

As a result of running this command, you will get a list of directories created just as in a Rails application. Here are the folders that gets created on running this command:

```
CHANGELOG CONTRIBUTORS INSTALL log Rakefile
script
config db LICENSE public README
vendor
```

We now change the config/database.yml to suit our database settings. By default, Radiant needs the user to create only a production database, however the user can choose to create a development and/or a test database too.

```
saurabh@laptop:~/radiant$ rake production db:bootstrap
```

Once we run this in our project directory, the bootstrap script will migrate the tables and add them to the database. At the end of that process, the script asks the user some questions necessary to setup and administer the application.

```
Create the admin user (press enter for defaults).
Name (Administrator): saurabh
Username (admin): admin
Password (radiant):
```

```
Initializing configuration.....OK
```

```
Select a database template:
1. Empty
```

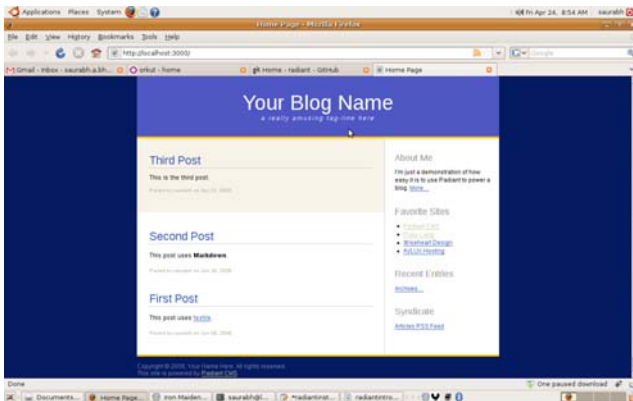
- 2. Roasters (a coffee-themed blog / brochure)
 - 3. Simple Blog
 - 4. Styled Blog
- [1-4]: 4

Creating Snippets....OK
 Creating Pages....OK
 Creating Layouts....OK
 Creating Page parts....OK

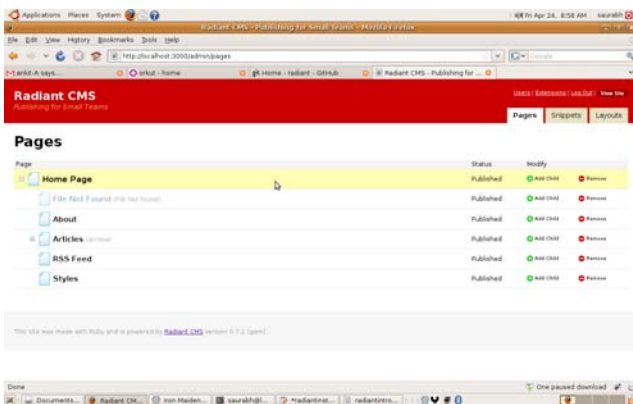
Finished.

Once we are through with these questions, we can start up the server in production mode and navigate to the `http://localhost:3000/` to see our application up and running.

```
saurabh@laptop:~/radiant$ script/server -e production
```



We are now ready to manage the content, so we navigate to `http://localhost:3000/admin/login` and login with the credentials we provided during the installation process. Once inside, we are presented with the administrator panel as shown in Figure 2.



Getting familiar with the Radiant terminology

The first tryst with Radiant might confuse you but once you understand the organization of components in Radiant, you can actually unlock the real power of Radiant, which is high modularity and re-usability of the components.

We will first get familiar with the small components that form a Page in Radiant, and then dive into how this page is formed using these sub-components.

Radius Tags

Radius tags are tags similar to HTML tags, but specific to Radiant CMS. They can be used anywhere inside Pages, Snippets or Layouts, except the names. Radius tags are intelligently built into four categories and that's what differentiates it from other tags. These are:

1. Content Tags: These tags refer to the different page attributes like title (`<r:title/>`), or content (`<r:snippet />`) and are mostly self enclosing in nature.
1. Tags that change Page context: These tags refer to a container tag and control from which page the contained tags get their content, e.g. `<r:parent>`
2. Conditional Tags: They start with `<r;if_` or `<r:unless_` and are container type tags. Only when the conditions defined in the tag are satisfied, the contained content or other elements are rendered.
3. Tags that work with collections: They are mostly the tags that do not fall under any other categories.

We can write custom radius tags in order to extend the radiant markup.

Snippets

As the name goes, snippets are small pieces of code, that perform different functions and can be reused. A snippet just needs to be created once. If you have to use some text across several different pages, reuse some pieces of code over different layouts, you can simply create a snippet and call it within that page or layout.

We can add a snippet as shown in the code sample below, by navigating to the snippet tab in the admin panel and clicking on "snippet". Here is an example of how you can create a reusable snippet:

Name: `table_of_contents`

Snippet:

```
<r:children:each limit="10" order="desc">
  <li><r:link /> </li>
</r:children:each></ul>
```

While saving, you have an option of keeping it as Markdown or Textile. Once created, this snippet can be called from within a page using a radius tag as follows:

```
<r:snippet name="table_of_contents"/>
```

Layouts

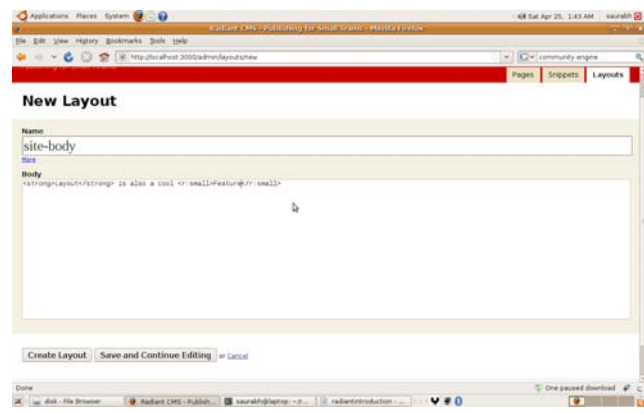
Layouts are like the body of HTML. They help us define how the page will look like and how the various elements will be arranged on the pages. Layouts are also reusable, just like

Snippets and can be included under other layouts too. The various portions of the layout like the content, can be written in any language we desire, apart from the radius tags, like HTML, XML, JavaScript. The screen to create a layout can be seen in Figure 3.



Pages

Pages follow a Parent – Child structure. A parent page can have many children pages and they can in turn have their children pages. This structure makes the site organization easier. Figure 4 shows how the page Admin looks like.



Pages are filled by snippets and layouts, yet they have their own set of attributes:

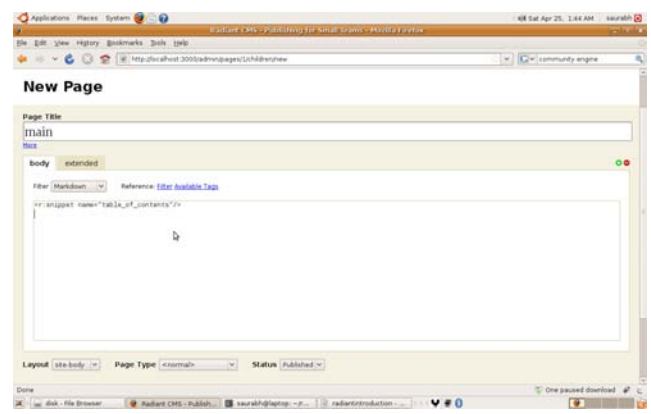
1. **Title:** This is the name of the Page in plain text. This shows up on the title bar of the browser when the page is displayed.
4. **Slug:** Slug is the link to the page being made. This link is generally taken automatically from the title of the page but it can be customized by clicking on the More link and then providing a custom text for it. Spaces are not allowed because these are parts of the URL.
5. **Breadcrumb:** Breadcrumb is for the user to see the navigation on the page. It is displayed in a format such that, a sequential trail is created starting from parent to the current page. It depicts the hierarchy of the page structure.

Other parts on the Page Admin are:

- **Filter:** This is helpful in selecting the content type for the contents and parts to be defined in a particular

page. The default options of Filter are Markdown, Smarty pants and Textile.

- **Page Types:** Page type helps us define the kind of pages for the site. Pages can be Normal, Blog Type and Archive. Depending upon the selection, they are showed in the final output. With File Not Found, a customized error page can be generated and used in case the defined error occurs on the site.
- **Status:** The Status of a Page depicts whether it has been saved or published by the user. Once published, the user will be able to see the page according to one's definitions.



Extensions

Radiant, just like any other CMS can be extended easily, when the default functionality of the radiant framework is not sufficient. There are several extensions available for Radiant, common ones being TinyMCE Editor and Latex for displaying mathematical formulaes and much more.

Conclusion

Radiant, as they say, is a no fluff CMS, provides us with a modular architecture which can be easily played around with. For a Rails programmer or a small rails team, Radiant makes it easier to extend and customize it to the specific needs of the user. Radiant also has a very active open source community.

Resources

- Radiant CMS Homepage - <http://radiantcms.org/>
- Radiant Wiki - <http://wiki.github.com/radiant/radiant>

DISCUSS: <http://railsmagazine.com/4/6>



Interview with Thomas Enebo

by Rupak Ganguly on August 1st, 2009

Rupak: What are the reasons behind the move to Engine Yard?

Thomas: Engine Yard is a very highly respected Ruby (and Rails) company. When they approached us they pointed out that JRuby could be better represented by a smaller more focused company than a larger one where Ruby is not a primary focus. They also were willing to include Nick Sieger as a Full-Time JRuby developer. So more resources and more focus. Very compelling...

Another benefit of Engine Yard will be agility in reacting to business opportunities. EY is not so large that they cannot quickly react to a changing market. If there is a good opportunity to offer commercial JRuby support or JRuby training, then I am confident that EY will act swiftly on those opportunities.

Rupak: You mentioned elsewhere the uncertainty after Sun's acquisition by Oracle. What related Sun initiatives do you feel may be in a similar situation?

Thomas: Big companies merging always have their share of tough decisions. Perhaps if two technologies overlap too much they may end up cancelling whole projects or partially merging those technologies together. I think overlaps is where most of the layoffs will come...but then again what do I know? I am not Oracle and I really have no idea what they might be interested in. Perhaps a particular overlapped technology at Sun will be more interesting than what they currently have at Oracle. Maybe some Oracle employees will get laid off. Maybe they will "double down" and more aggressively develop that part of the business.

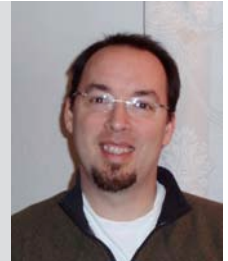
It is difficult to guess on how a large company will make decisions. I certainly don't want to make any predictions since I would hate for them to come true. Especially, since I know many engineers at Sun and I would hate to jinx their chances.

Rupak: What type of support will you get from Engine Yard that wasn't available before? How would that affect the future of JRuby?

Thomas: I touched on this a little bit in 1, but I will add a little more...

EY is growing the full-time developers on our team to three. So 50% more JRuby developer goodness. On top of that I am sure that we will be getting help on making better documentation and at least some level of support for JRuby users. The fact that we are joining a Ruby company is important since it also means we will have access to Ruby experts and people who understand how to support Ruby in a cloud envi-

Thomas Enebo has been working with Java since 1997 and co-leading the JRuby project since 2003. Thomas is working to make JRuby a piece of software that will capture the hearts and minds of Ruby and Java developers everywhere. His goal is to make JRuby the best possible Java Virtual Machine (JVM) implementation of the Ruby language and to make the JVM and the Java platform the best possible host for all languages.



ronment. Also access to Rails committers like Yehuda Katz. And these last few points should remind us that Engine Yard has a great reputation in the Ruby and Rails community. This will help expand our reach within the Rails community. I hope this announcement will help give Ruby users another reason to consider investigating JRuby.

Rupak: EngineYard started to offer JRuby hosting recently. Will you be part of that or is your time dedicated to further JRuby development?

Thomas: We will still be working on JRuby itself, but we will be helping as needed with any discovered problems or missing functionality in JRuby. Largely, the problems that Engine Yard may run into for a hosting service are the same problems that other JRuby users will run into.



Session on JRuby by Thomas Enebo and Nick Sieger from Sun Microsystems



WindyCityRails

An annual gathering for all who are passionate about Ruby on Rails.

Saturday, September 12, 2009
Chicago, Illinois



windycityrails.org

Rupak: What things should the community expect from the JRuby team after this move? What are your expectations or needs from the community?

Thomas: Compatibility and performance improvements should always be expected from our development activities... but I think this move is going to give us more focus on sanding and polishing some of the rough edges of JRuby. Documentation, extra steps that are not required by C Ruby, etc...

I am hoping this move will help expand the Ruby and JRuby communities and this in turn will help us get more bug reports and success stories; which in turn will help expand the Ruby and JRuby communities...and so on :)

Rupak: Would you like to add anything else for our readers?

Thomas: In the next year we also plan on courting Java developers more by making improvements which makes it easier for Java developers to start using Rails without having to “start over” with their codebase.

I think it is important to grow the size of the Ruby and Rails communities and Java is the 800 pound gorilla in the room. Getting more Java programmers using Ruby/Rails will also start making Ruby and Rails even more acceptable technologies in bigger companies. This in turn should help Rubyists find more jobs.

JRuby is not only a great Ruby implementation it is also a gateway technology and we should concentrate on that side of JRuby this year.

DISCUSS: <http://railsmagazine.com/4/7>

Audience at Ruby Kaigi



Oracle Tips and Tricks

by Greg Donald

Oracle database software's near-zero cost in academia result in ubiquitous usage. Pairing Rails with Oracle isn't presently void of issues, but there are no real show stoppers. Here are a few things I've ran into and how I worked around them.

Oracle's Missing Time Field

Oracle doesn't have a time field like other popular databases. You may very well define a `:time` type in an `ActiveRecord::Migration`, but Oracle will give you a date field instead. The date field will hold a time value most of the time, but not always.

A sort of "gotcha" occurs when you get a record with the time portion set to midnight. In this special case Oracle doesn't return any time value at all in this special case, instead it just returns the date part of the record. On the Ruby/Rails end you get back a `Date` class instead of a `Time` class. This means your time field will not have `min` or `hour` methods. The solution is coercion, using `to_time`:

```
foo = Foo.find( :first )
foo.my_time_field = foo.my_time_field.to_time
```

After that your date field will contain a time that's set to midnight (luckily this matches our special case from above exactly). Another workaround is to rescue the exception with a zero value:

```
foo.my_time_field.hour rescue 0
foo.my_time_field.min rescue 0
```

Keep session data stored in Oracle simple

The Oracle-Ruby database driver has some issues with storing complex data in a session. I had a scenario where I needed to store objects in a session temporarily while waiting to put their parent data in the database first. Something like this would cause the issue:

```
foo = Foo.new( params[:foo] )
session[:foos] << foo
```

I suspect the cause is the fact that an `ActiveRecord` object owns a database handle, and when you accumulate several of them, threading issues arise. I will admit I did not pursue it further once I found a workaround.


The solution was to not store the complex data in the session at all, but instead only store a session variable pointing to the data:

```
foo = Foo.create( params[:foo] )
session[:foos] << foo.id
```

This solution works fine but has the side effect of allow-

ing orphaned data to appear in the database if a session to browser connection is ever lost.

I will also mention this issue never bubbled up until my app was deployed to a 64-bit machine. In a 32-bit environment it seems I can store all the `ActiveRecord` objects in a session that I want to.



Greg Donald is a career software developer currently working in computational genomics research at Vanderbilt University Medical Center in Nashville, TN. Greg has a wife, three children, and five cats. Greg is a member of both the Free Software Foundation and the Linux Foundation and prefers to use open-source software whenever possible. In addition to writing new Ruby and Rails code for fun and profit, Greg also enjoys shredding along to his favorite Slayer tunes on his 6-string <http://slayer.net/>.
Web site: <http://gregdonald.com/>

Learn to purge Oracle's recycle bin

Rails migrations allow the destroying and rebuilding of a project database on a whim.

```
rake db:migrate VERSION=0; rake db:migrate
```

A "gotcha" occurs after running these commands several dozen times or more. Newer versions of Oracle do not actually drop constraints. Instead Oracle saves them to the Oracle recycle bin. Over time they build up and cause otherwise speedy migrations to take longer and longer to run.

The solution is to dump Oracle's recycle bin periodically. Add this to the end (and beginning too if you want) of your `ActiveRecord` migrations:

```
execute 'purge recyclebin'
```

Oracle's Sequence Cache

By default Oracle caches requests for new sequences in blocks of 20. This might become a problem if you need numbers without gaps as you will get 1, 2, 3... and then later 23, 24, 25... To fix it I just added this to the appropriate `ActiveRecord::Migration`:

```
execute 'ALTER SEQUENCE FOO_SEQ NOCACHE'
```

DISCUSS: <http://railsmagazine.com/4/8>

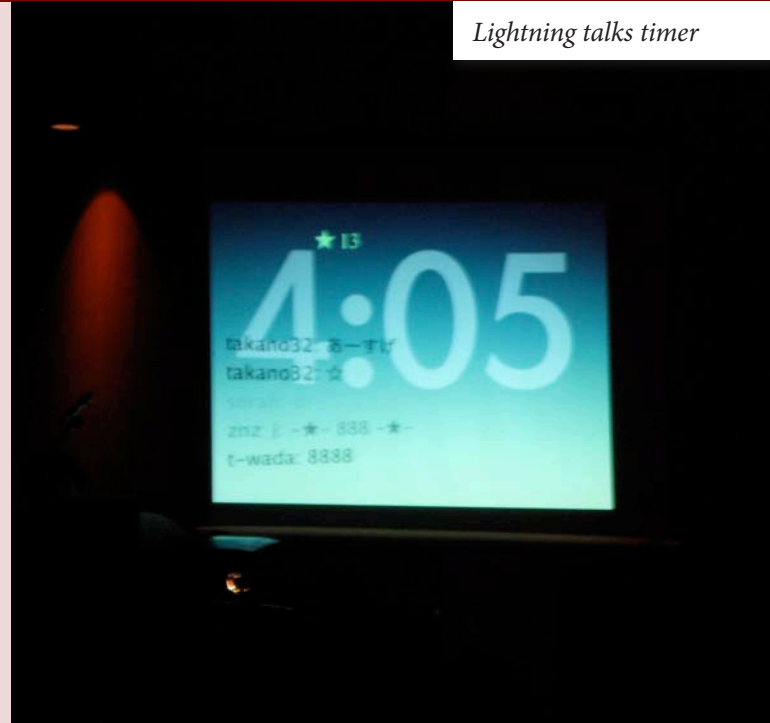
Ruby Kaigi Rails Magazine Exclusive Coverage

RubyKaigi or "Ruby Conference" is the largest Ruby conference in Japan, where Ruby was born. Tens of professionals including Ruby-core committers are giving their talks on Ruby in different fields and perspectives.

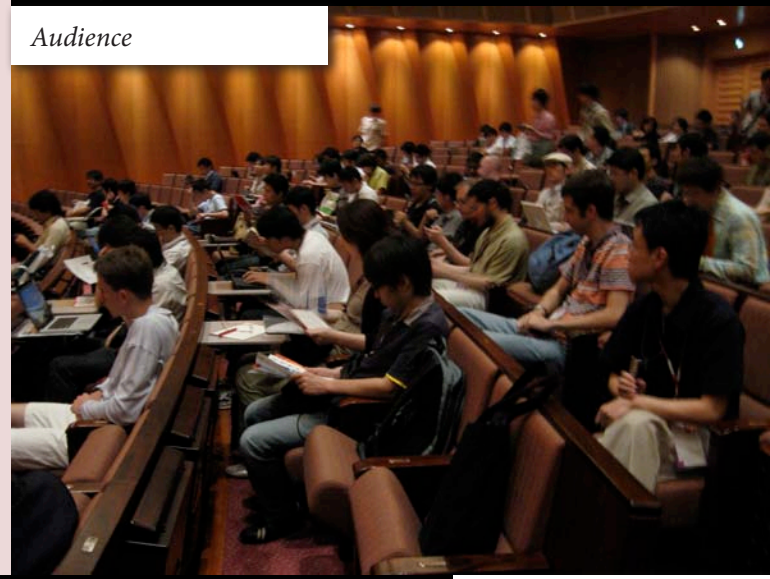
RubyKaigi 2009 took place in Tokyo, between 17-19th of July.

The speakers list of this year included Matz, Koichi Sasada, Katz, Tom Enebo, Nick Sieger and many other professionals.

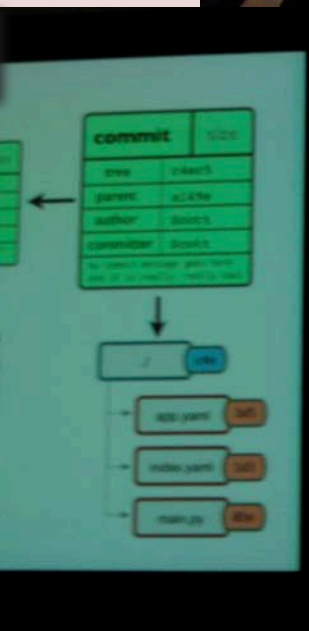
The official site is <http://rubykaigi.org/2009/en>



Audience



Muhammad Ali, eSpace
Session on Neverblock



Scott Chacon, Github
Session on Git and Github



RubyKaigi 2009 Roundup

by Ehab El-Badry

RubyKaigi 2009 (Tokyo, July 17–19)

‘The Japanese have a name for their problem: Galápagos Syndrome.’ As I was flying back from Tokyo, I had to admit, as I folded the newspaper and stowed it into the pouch in front of me, that was the same feeling I had. Sure the article was about cell phones and how many ‘species’ of such advanced technology have developed and evolved so fantastically divergent from the rest of the world...

But Ruby is not like that! It’s one of those species that was brave enough to leave the sanctuary of its own island to venture off into other worlds — even leaving an impact on them. If you’re reading this, that’s probably what you want to believe. But to be honest, the kaigi (or ‘meeting’ or ‘conference’ in Japanese) did show some signs of the isolation syndrome, as much as it did show globalization. In the following lines, we will briefly cover the largest Ruby meet-up in Ruby’s native homeland and muse over some of its highlights.

‘Change!’ is in the Kaze

On the first day, during registration, a cheerful volunteer handed me my name-tag and conference package. The promotional knapsack was all black, save for the red words written on it that desperately screamed ‘CHANGE!’. I felt relieved; apparently the organizers also felt that the syndrome must be eradicated, so much so that they made it the slogan of this year’s conference and saw it as a goal to be achieved – in fact the idea of “Change” was one of the pivotal points in the keynote speech.

It’s true that change was indeed in the works. In the fourth conference since RubyKaigi’s existence, foreign speakers (such as ourselves) were invited, making it a multilingual conference for the first time. As one of the posters proudly boasted, there were speakers and attendees from five continents who had presented around a quarter of the talks. It was refreshing to see how all the sessions – both in Japanese and English – were very well attended by the ~300 individuals there.

Organization, Organizers, and Volunteers

In a city that never sleeps, and works by the clock, the organization level and punctuality in the conference was nothing short of what one would expect from the Japanese. The conference was very well coordinated and organized from sessions, to bento lunches and refreshments, to parties and socialization. Each day, the schedule was split into three time slots to contain multiple sessions in each, with talks going on in parallel to squeeze in all the ~60 sessions in the three days available. The first two days also had a series of lightning talks that were composed of extremely short presentations for tens of people

to display their latest work, speaking in the fastest Japanese (or English, as in our case) physically possible. Even though they were too fast for translation, it was interesting to watch ideas fly by with a bit of manga-like humour.

It is a pity though that many important talks were in Japanese, from the opening, all the way through the keynotes, down to the closing. There was best-effort live translation via a projector, but that didn’t really do justice to the talks and there were periodic translation outages. I must admit though, the organizers and volunteers did try to do their best, but the sheer number of attendees and talks was simply overwhelming. At this moment, I really must mention Leonard Chin, the alleged mastermind behind the organizing team. All the volunteers were helpful and energetic as well, despite the language barrier — after all we all spoke Ruby!

Also, keeping true to the Japanese fondness of having a good drink after a hard day’s work, the organizers orchestrated a pre-conference party and two other receptions throughout the conference. Even though it’s not as fun for non-drinkers like us, it still presented a great opportunity for all the participants to get to know each other better.

A True Ruby Mecca

As hundreds of wild-haired, glasses-donning, computer otaku flocked to the main Hitotsubashi conference hall, one was bound to feel a shiver down his spine: this is, after all, where Ruby was born. Numerous beehive discussions were taking place and there was even in one corner a donation box dedicated to gathering funds to get “_why” to come as well in some unforeseen, yet hopeful, future. No, this is not just where Ruby was born, but where it continues to live and grow. The hundreds in attendance were musing over different ideas and dreams to as what Ruby will grow to be someday when, suddenly, they fell into a trance and were all ears as the keynote speaker came forth to have his say. Their reaction was just like that of those looking up to the pope as he steps into his papal



Map showing where foreign participants are from

balcony, eager to listen and take in what is to come.

Just in case you haven't guessed yet, the keynote speech was delivered by the designer of Ruby, Yukihiro Matsumoto (aka Matz). Where else but in Japan would a computer scientist be a folk hero, a celebrity known by even the normal population, and even the subject of tabloid news?

Again, a pity that the keynote was in Japanese, but Matz



Ehab El-Badry is COO at eSpace and has a knack for project management of web projects and was once a sysadmin in his previous life. And because work is not everything in life, he enjoys traveling to exotic places around the world, and learning foreign cultures and languages such as Japanese.

was a really good presenter and he easily grasped the attention of the entire hall. Here are some highlights:

- We had dreamt that the 21st century would have been better than this – but in reality we are living in dark times. Yet, this is the golden age for programming – especially with technologies like Ruby, and the likes of other very powerful and clean languages.
- I just recently got an award from a minister for my creating Ruby. I had to wear a jacket during the meeting, so I borrowed one. Likewise, I couldn't have made Ruby without the help and contribution of others. All bug reports, ideas and contributions were extremely helpful and, even though I may not buy you dinner, I have to share this award with you all.
- The Ruby community is changing a lot and it is very evident from the diversity of attendees and speakers here at RubyKaigi.

- As in Aesop's 'Sour Grapes', I am not immune: I look at other languages and love to complain. But people should not misinterpret this to mean that the stance of Ruby is to be against all others – this is just a matter of my own taste.
- It's important to not turn your back on the truth and always make self-justifications. I admit that Ruby has many faults, and, although it may take a long time to fix them, they will undoubtedly be resolved. Responsibility is the key towards doing this right.
- I remember when I was a kid, my dad gave me a knife to sharpen my pencils with; every boy at the school had one of these. It is unimaginable these days that kids can go to schools with knives, yet in our time we never used them to stab each other. This is responsibility.
- I am experimenting with some interesting stuff for Ruby like O(1) bitmap marker, GC for symbols, better Date implementation, keyword arguments and many more improvements.
- To wrap things up, Ruby is fair — be responsible and Ruby trusts you to be so. We need to remember that not only is Ruby agile, but it is also fragile.

Highlights from the Sessions

The sessions varied widely in scope, from talks focused on integrating Ruby with other technologies like Asterisk and iPhone Cocoa, to talks about building scalable Ruby applications in certain domains, to talks about specific libraries and services. Moreover, as one being at the homeland of the Ruby core team would expect, there were several sessions talking about the internals of Ruby and the future of the Ruby virtual machine.

The kickoff session was from Github's Scott Chacon, who talked about Git and how it can be used to improve Ruby developers' performance, especially those working on open source projects. The presentation was titled "Using Git and



Muhammad and Matz

GitHub to Develop One Million Times Faster*”, complete with the fine print on how results may vary. It really did display a great deal of interesting features and apparently most of the audience were using Git or are planning to migrate to it — we know we are!

Ruby’s viability in communication software was stressed a lot by the Adhearsion talk given by Jason Goetze, which showed how Ruby can drive Asterisk installations with all the ease and flexibility of Ruby, without sacrificing the power of Asterisk. One example used was the automation of call centers. Another very interesting example was *twittervotereport.com*, which helps users report, via phone, real time complaints on vote ballots using voice menus and keystrokes. The demos shown stressed the mix of power and simplicity in Adhearsion.

James Edward’s talk, “How Lazy Americans Monitor Servers”, highlighted how Ruby can be used in long running applications like monitoring systems. He showcased Scout, which is a plugin/recipe based monitoring system with very low overhead. He was able to use the power of Ruby to build a very flexible system and utilize multiprocessing to the fullest to add the robustness needed from long running programs.

Ilya Grigorik from *igvita.com* presented how Ruby can be useful even for demanding problems like the C10K challenge, which basically states that current server hardware should be able to handle a very large number of concurrent connections — namely 10,000. Ilya reviewed some techniques and recommendations for shaping Ruby up to the challenge, and then he demonstrated their own EM-Proxy as an example.

Yehuda Katz’s talk on “Making Rails 3 a Better Ruby Citizen” focused on the new changes happening to Rails internals to make it more Ruby-like. He concentrated on the use of common Ruby idioms and provided examples to show that they are as powerful as the current Rails way but they are cleaner and expose much nicer interfaces. Another issue was the interconnection between different Rails components like ActionCon-

troller and ActionView. The idea with Rails 3.0 is to minimize the areas where these components interact and keep it to a few publicly known methods, thus allowing a third party to provide replacements easily. Even internal components of active record, like validations or life cycle management, are now separated from each other and can be used, or even swapped, independently.

eSpace’s Mohammad Ali presented NeverBlock and the talk focused on Ruby concurrency models and how they can optimize Ruby for high I/O performance. He presented NeverBlock as an option that can optimize Ruby’s I/O while maintaining program readability. A case study was given where a new back-end that uses NeverBlock was written for a Thin server and was able to largely outperform the original back-end, especially for large data transfers.

While there were nice comments and jokes embedded in most presentations, it was Aaron Patterson who went the extra mile in entertaining the audience. Dressed like a Japanese pirate, he gave a mixed Japanese and English talk and was assisted by a kunoichi (or female ninja) who guided the audience whether to ‘laugh’ or ‘applause’ throughout the presentation using cue cards. His lecture, a remake of “Journey Through a Pointy Forest: XML Parsing in Ruby”, was about Nokogiri and XML handling in Ruby. For added laughs, he poked fun at his Enterprise project which converts Ruby code to XML in order to make Ruby more enterprise ready. ;)

Retrospective

All in all, it was an experience we’d definitely want to go through again and we highly recommend it. All the presentations were top-notch, entertaining and greatly informative, and every single one of them ended with a great number of questions, signaling how relevant they were to the audience.

Sure there were barriers, and they really did show. There were many things we never heard of, or were not aware of, coming from the outside world. At the beginning, it was frustrating not just to us, but to other “foreign” Rubyists as well, but throughout the conference things loosened up and the barriers dissolved. No one says that they don’t exist, actually everyone fully acknowledges they do, but if there is one thing the kaigi actually accomplished, it is exactly what it set out to do: inflict change.

To be able to rub shoulders with people like Matz, Koichi, and numerous others, to discuss future road plans, to expose — and be exposed to — the latest in what there is to know from the core team and others, is beyond anything a Rubyist can ask for. If you are able to cross that thinning language barrier and are willing to show off some fluent Ruby, then after a whiff of wasabi you’ll know this is as real as it gets.



Session on Nokogiri by Aaron Paterson

DISCUSS: <http://railsmagazine.com/4/10>



Interview with Matz

by Muhammad Ali on June 20th, 2009

Oldmoe: Hello Matz, we'd like to start with the question, What are we going to see with Ruby 1.9.2?

Matz: An even more stable 1.9.x, 1.9.0 was released 2 years ago but not really that stable (much more like a technology preview) After one more year of working we released 1.9.1 which was delayed a bit. The community assigned an official release manager for 1.9.x series to streamline the releasing process. While 1.9.1 is supposed to be stable we aim to make 1.9.2 even more stable and more compatible with 1.8.7. Lots of bugs got fixed. Unicode processing is more stable and faster than 1.9.1.



Yukihiro Matsumoto (aka Matz) is a Japanese computer scientist and software programmer best known as the chief designer of the Ruby programming language and its reference implementation, Matz's Ruby Interpreter (MRI).

Oldmoe: What is the status of the Ruby GC in 1.9.2? Some of the latest commits indicate a generational collector is being experimented with.

Matz: Author Nari is assigned GC experimentation. He wrote a patch that provides generational support for special node objects. Those are allocated on a different heap that is

scanned less frequently. Ko1 recently changed the byte code structure to use less node references which should also make GC runs finish faster. This is most useful in applications that load lots of classes and modules, e.g. Rails.

Oldmoe: What else is planned for 1.9.2?

Matz: No syntax changes planned. A notable thing is that we relaxed limitations on time periods for the Time classes, we are no longer limited by UNIX time periods. And the good thing is that we still use UNIX time for performance when the time object lies within its boundaries.

Oldmoe: And what is planned after 1.9.2?

Matz: Work should start on Ruby 2.0, I want to improve on the code scalability aspect of the language design. May be introducing a new name space management that helps with huge projects. This tough to implement efficiently so we need to investigate this.

Oldmoe: Do you code in other languages? Python?

Matz: I write several lines of code to experiment with it. It has a rich set of libraries. But the design is drastically different.

Oldmoe: Are you considering any new programming models to incorporate into Ruby?

Matz: It might be good to utilize multiple processes for multicore processing. Also Ruby provides very little in IPC

29 Steps. "We love the web ... and Rails."

We are a startup company who specializes in Ruby, Rails and the web.

Visit us at <http://29steps.co.uk>

Ruby . Ruby-on-Rails. Iphone. Web Design. Application Development. Audit. User Interface Design. Ruby-On-Rails training.



primitives. It would probably be good to have a multiprocessing implementation.

Oldmoe: Shouldn't the standard library go on diet?

Matz: Maybe but it's a long term process, we need to emphasize compatibility between minor versions. Not the case with Ruby 2.0 which should have less bundled libraries.

Oldmoe: Do you have any design regressions? Things you wish were done differently?

Matz: Local variable scoping. When the scope is determined lexically but other scopes might affect it in place. Can cause bugs that are quite difficult to find.

Oldmoe: How do you see the Ruby community lately

Matz: I still think the Ruby people are 'nice' people and I want them to be fair. There should be no dictatorship. Everyone should be rational with each other.

Our challenge is how to embrace the people from the rails community and we need to integrate them into the culture of the Ruby language.

Oldmoe: Shouldn't the Ruby culture evolve to adapt them?

Matz: The Ruby culture should evolve all the time. 10 years ago we were more influenced by Perl programmers. Recently functional language programmers are leading the cultural change.

Oldmoe: Aren't you afraid that one day the PHP programmers will be leading the cultural change?

Matz: I hope we can integrate them as well.

Oldmoe: Any exciting news for Ruby that you can share with us?

Matz: The Japanese government gave a grant for making Ruby and other Japanese software projects faster. For them to be suitable for HPC by maybe utilizing some sort of JIT compiler or a AOT compiler. We aim to build NumPy, SciPy equivalents for Ruby to make Ruby suitable for areas beyond web development.

Oldmoe: What about embedding Ruby?

Matz: No budget is set yet but it is on the Radar so it has to progress on a community basis as of currently.

Oldmoe: When you wrote Ruby did you see that it will make this revolution in web development?

Matz: No, I have not seen that, it really exceeded my expectations. Feels like living a dream, one that you don't want to wake up from.

Muhammad Ali is co-founder of, and CTO at eSpace Technologies (www.espace.com.eg), a software development firm that is located in sunny Alexandria, Egypt and is specialized in Ruby on Rails.

He had been playing with a diverse set of languages from C to Java and JavaScript for eleven years when he finally entered into a steady relationship with

Ruby and Rails four years ago. Oldmoe has contributed to many open source projects and is particularly interested in researching software concurrency models.

Blog: oldmoe.blogspot.com



Oldmoe: With the new Ruby implementations appearing everyday aren't you afraid to lose your control of the language?

Matz: Currently they respect my design and even if they take it beyond that I will be a fork and not Ruby itself (maybe Ruby++) so I am not worried that much.

Oldmoe: Thank you Matz for your time.

DISCUSS: <http://railsmagazine.com/4/11>



Interview with Koichi Sasada

by Muhammad Ali on June 20th, 2009

Oldmoe: Hello Koichi, let's start by asking what are you planning for Ruby 1.9.2?

Koichi: I am mostly interested in the core. Optimization, debugging and profiling support. We are now trying to define the API to expose the internal structure. They were open in 1.8 but we had to close them and we need to expose C level API functions. I am mostly interested in cleanly exposing the internal structure for a tracing API. If we conclude the design soon it will make it to 1.9.2.



Koichi Sasada is the creator and current maintainer of YARV, the official Ruby Interpreter for Ruby 1.9. He is also running the Sasada laboratory at The University of Tokyo, researching Ruby and virtual machines.

Oldmoe: What is the current state of GC optimizations?

Koichi: It is hard to implement a generational garbage collection of the current MRI. Mostly because of write barrier issues. Author Nari is experimenting with a semi generational garbage collection that stores nodes in a different heap. This leads to the GC not checking for code very often. But currently we store inline cache in specially managed heaps by the VM and we do not expose them to the GC thus making this part more efficient. We will most likely use that and roll back the semi generational garbage collector.

Oldmoe: Matz mentioned that Ruby is being funded for use in HPC, how will the VM adapt?

Koichi: Yes, currently Ruby has received funding to improve its use in HPC. These guys usually use Fortran and we hope to reach a state for the first 5 years which will enable Ruby to be viable in the HPC area at least for effective prototyping. This is mostly an academic effort so we will need to publish many papers. We need to balance the academic efforts with providing practical implementations.

Oldmoe: A tracing API for use in a tracing optimizer?

Koichi: Well, the new tracing API is mainly geared towards debugging and profiling rather than optimization at this stage. Ultimately we want to build an optimizing JIT.

Oldmoe: And what about RICSIN? Will it come to 1.9.2?

Koichi: I want to move C extensions to RICSIN. Which is more efficient because RICSIN calls C methods directly. It might not be included in 1.9.2 due to time constraints. Though I can sneak it in via a single instruction modification to the VM without other committers noticing, except Matz, I will probably do that!

Oldmoe: What about the current status of the fiber implementation?

Koichi: Using getcontext/setcontext should help fibers be much faster. It should help some clever library writers to use it. If someone has idea to improve fiber features please tell us. We basically wrote the fibers for enumerators but not for scalable I/O. Seeing them used for scalable I/O is interesting.

Oldmoe: What about Multi Virtual Machines (MVM)?

Koichi: My goal of MVM is to find an efficient way to communicate between several VMs in the same process. This

Muhammad and Koichi





should also enable parallel computing in a single process for CRuby. It is currently lagging the current trunk. It needs some work to enable it to use C extensions.

Oldmoe: What about a multi process approach to MVM?

Koichi: We are not considering it right now. But ultimately we want the API to seal the actual implementation which might decide to fork a new process or a new thread or even start a new process on a new machine across the network.

Oldmoe: What are your plans for the future of the Ruby VM?

Koichi: I believe some compilation can be done. Either JIT or AOT compilation. I am considering a tracing optimizer. It should enable Ruby to gain optimizations like those that happened for Firefox and Safari Javascript engines. Ultimately we don't want to hand code a very complex VM. Whenever we can automate the optimizations it will be better as we need to keep the core of the VM very simple.

Oldmoe: What about facilities for writing optimizers in Ruby?

Koichi: I have made a proposal to Matz to provide a `to_s` method to procs but we are still considering it. This should provide the ability of doing runtime optimizations in Ruby. I would say that we want to have more clever ideas to improve performance without sacrificing the dynamic features of Ruby.

Oldmoe: Thank you Koichi for your time.

DISCUSS: <http://railsmagazine.com/4/12>

Rails Magazine Team

Olimpiu Metiu

Editor-in-chief

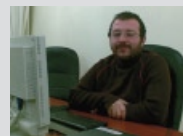
<http://railsmagazine.com/authors/1>



Khaled al Habache

Editor

<http://railsmagazine.com/authors/4>



Rupak Ganguly

Editor

<http://railsmagazine.com/authors/13>



Mark Coates

Editor

<http://railsmagazine.com/authors/14>



Carlo Pecchia

Editor

<http://railsmagazine.com/authors/17>



Starr Horne

Editor

<http://railsmagazine.com/authors/15>



John Yerhot

Editor

<http://railsmagazine.com/authors/2>



Bob Martens

Editor

<http://railsmagazine.com/authors/16>



Call for Papers

Top 10 Reasons to Publish in Rails Magazine

1. **Gain recognition – differentiate and establish yourself as a Rails expert and published author.**
2. Showcase your skills. Find new clients. Drive traffic to your blog or business.
3. Gain karma points for sharing your knowledge with the Ruby on Rails community.
4. Get your message out. Find contributors for your projects.
5. Get the Rails Magazine Author badge on your site.
6. You recognize a good opportunity when you see it. Joining a magazine's editorial staff is easier in the early stages of the publication.
7. Reach a large pool of influencers and Rails-savvy developers (for recruiting, educating, product promotion etc).
8. See your work beautifully laid out in a professional magazine.
9. You like the idea of a free Rails magazine and would like us to succeed.
10. Have fun and amaze your friends by living a secret life as a magazine columnist :-)



Call for Artists

Get Noticed

Are you a designer, illustrator or photographer?

Do you have an artist friend or colleague?

Would you like to see your art featured in Rails Magazine?

Just send us a note with a link to your proposed portfolio. Between 10 and 20 images will be needed to illustrate a full issue.

Join Us on Facebook

<http://www.facebook.com/pages/Rails-Magazine/23044874683>

Follow Rails Magazine on Facebook and gain access to exclusive content or magazine related news. From exclusive videos to sneak previews of upcoming articles!

Help spread the word about Rails Magazine!

Contact Us

Get Involved

Contact form: <http://railsmagazine.com/contact>

Email: editor@railsmagazine.com

Twitter: <http://twitter.com/railsmagazine>

Spread the word: <http://railsmagazine.com/share>

Sponsor and Advertise

Connect with your audience and promote your brand.

Rails Magazine advertising serves a higher purpose beyond just raising revenue. We want to help Ruby on Rails related businesses succeed by connecting them with customers.

We also want members of the Rails community to be informed of relevant services.

Visit Us

<http://RailsMagazine.com>

Subscribe to get Rails Magazine delivered to your mailbox

- Free
- Immediate delivery
- Environment-friendly

Take our Survey

Shape Rails Magazine

Please take a moment to complete our survey:

<http://survey.railsmagazine.com/>

The survey is anonymous, takes about 5 minutes to complete and your participation will help the magazine in the long run and influence its direction.

