Camping: Going off the Rails with Ruby

Adventures in creative coding for people who should know better

The weasel words

* This presentation contains code
* That code is probably broken
* If that bothers you - fix it
* That's called a learning experience

Who are these lunatics?

Eleanor McHugh
[Games With Brains]

She does real-time systems

Romek Szczesniak
[Telnic Limited]

He does security

Alright, but what are they doing here?

* Ruby
* WEBrick
* Camping
* OpenSSL
* but no Rails...

No Rails?

* That's right, we don't use Rails
* But we do use Ruby
* And we do write web applications
* So how is that possible?

Camping!!!

* That's right, we use Camping
* It's by Why The Lucky Stiff
* It's cool
* It's really cool
* It's so damn cool you'd have to be mad not to use it!!!

It's this simple!

```
%w[rubygems active_record markaby metaid ostruct].each {|lib| require lib}
 module Camping;C=self;module Models;end;Models::Base=ActiveRecord::Base
 module Helpers;def R c,*args;p=/\(.+?\)/;args.inject(c.urls.detect{|x|x.
 scan(p).size==args.size}.dup){|str,a|str.gsub(p,(a.method(a.class.primary_key
 )[]rescue a).to_s)};end;def / p;File.join(@root,p) end;end;module Controllers
 module Base;include Helpers;attr_accessor :input,:cookies,:headers,:body,
 :status,:root;def method_missing(m,*args,&#38;blk);str=m==:render ? markaview(
 *args,&#38;blk):eval("markaby.#{m}(*args,&#38;blk)");str=markaview(:layout){str
 }rescue nil;r(200,str.to_s);end;def r(s,b,h={});@status=s;@headers.merge!(h)
 @body=b;end;def redirect(c,*args);c=R(c,*args)if c.respond_to?:urls;r(302,'',
 'Location'=&gt;self/c);end;def service(r,e,m,a);@status,@headers,@root=200,{},e[
 'SCRIPT_NAME'];@cookies=C.cookie_parse(e['HTTP_COOKIE']||e['COOKIE']);cook=
 @cookies.marshal_dump.dup;if ("POST"==e['REQUEST_METHOD'])and %r|\Amultipart\
 /form-data.*boundary=\"?([^\";,]+)\"?|n.match(e['CONTENT_TYPE']);return r(500,
 "No multipart/form-data supported.")else;@input=C.qs_parse(e['REQUEST_METHOD'
 ]=="POST"?r.read(e['CONTENT_LENGTH'].to_i):e['QUERY_STRING']);end;@body=
 method(m.downcase).call(*a);@headers["Set-Cookie"]=@cookies.marshal_dump.map{
 |k,v|"#{k}=#{C.escape(v)}; path=/"if v != cook[k]}.compact;self;end;def to_s
 "Status: #{@status}\n#{{'Content-Type'=&gt;'text/html'}.merge(@headers).map{|k,v|
 v.to_a.map{|v2|"#{k}: #{v2}"}}.flatten.join("\n")}\n\n#{@body}";end;def \
 markaby;Class.new(Markaby::Builder){@root=@root;include Views;def tag!(*g,&#38;b)
 [:href,:action].each{|a|(g.last[a]=self./(g.last[a]))rescue 0};super end}.new(
 instance_variables.map{|iv|[iv[1..-1].intern,instance_variable_get(iv)]},{})
 end;def markaview(m,*args,&#38;blk);markaby.instance_eval{Views.instance_method(m
 ).bind(self).call(*args, &#38;blk);self}.to_s;end;end;class R;include Base end
 class NotFound&lt;R;def get(p);r(404,div{h1("#{C} Problem!")+h2("#{p} not found")
 });end end;class ServerError&lt;R;def get(k,m,e);r(500,markaby.div{h1 "#{C} Prob\
 lem!";h2 "#{k}.#{m}";h3 "#{e.class} #{e.message}:";ul{e.backtrace.each{|bt|li(
 bt)}}})end end;class&lt;&lt;self;def R(*urls);Class.new(R){meta_def(:inherited){|c|
 c.meta_def(:urls){urls}}};end;def D(path);constants.each{|c|k=const_get(c)
 return k,$~[1..-1] if (k.urls rescue "/#{c.downcase}").find {|x|path=~/^#{x}\
 \/?$/}};[NotFound,[path]];end end end;class&lt;&lt;self;def escape(s);s.to_s.gsub(
 /([^ a-zA-Z0-9_.-]+)/n){'%'+$1.unpack('H2'*$1.size).join('%').upcase}.tr(' ',
 '+') end;def unescape(s);s.tr('+', ' ').gsub(/((?:%[0-9a-fA-F]{2})+)/n){[$1.
 delete('%')].pack('H*')} end;def qs_parse(qs,d='&#38;;');OpenStruct.new((qs||'').
 split(/[#{d}] */n).inject({}){|hsh,p|k,v=p.split('=',2).map{|v|unescape(v)}
 hsh[k]=v unless v.empty?;hsh}) end;def cookie_parse(s);c=qs_parse(s,';,') end
 def run(r=$stdin,w=$stdout);w&lt;&lt;begin;k,a=Controllers.D "/#{ENV['PATH_INFO']}".
 gsub(%r!/+!,'/');m=ENV['REQUEST_METHOD']||"GET";k.class_eval{include C
 include Controllers::Base;include Models};o=k.new;o.service(r,ENV,m,a);rescue\
 =&gt;e;Controllers::ServerError.new.service(r,ENV,"GET",[k,m,e]);end;end;end
 module Views; include Controllers; include Helpers end;end
```

Why?

* For fun
* For profit

* For the satisfaction of knowing exactly how your application works
* For the look on your boss's face when he reads the documentation

Earlier...

* But let's not get ahead of ourselves
* First we want to take you on a journey
* A journey back in time
* A journey back to...

January 3rd 2006

* Location: The Secret Basement Lair™ of Captain IP and The DNS avengers

* Their task: to launch a new Top Level Domain which DOESN'T RESOLVE MACHINE ADDRESSES?!?!?!

* Their resources? Erm....

Ruby to the Rescue

* It's easy to learn
* It's quick to code in
* It's pleasing to the eye
* It's fun!

You keep saying that

* Yes!!!
* Fun makes for better coders
* Better coders write good code
* Good code stands the test of time
* If coding isn't fun YOU'RE USING THE WRONG TOOLS!!!!

The console jockeys

* let's write a menu driven calculator
* output: puts(), print()
* input: gets(), termios library
* old-fashioned and unattractive
* termios is fiddly

A simple calculator

```
#!/usr/bin/env ruby -w
require 'termios'

$total = 0
$menu_entries = [['+', "Add"], ['-', "Subtract"], ['*', "Multiply"], ['/', "Divide"], ['c', 'Clear'], ['q',"Quit"]]

$commands = $entries.inject([]) { |commands, entry|
        commands << entry[0]
        }

$captions = $entries.inject([]) { |captions, entry|
        captions << entry[1]
        }

loop do
        puts "\nSimple Calculator\n"
        entries.each { |entry|      puts "#{entry[0]}. #{entry[1]}\n"      }

        t = Termios.tcgetattr(STDIN)
        t.lflag &= ~Termios::ICANON
        Termios.tcsetattr(STDIN,0,t)

        begin
                action = STDIN.getc.chr
        end until $commands.member?(action)

        exit() if action == $commands.last
        action =  $commands.index(action)
        puts "\n#{$captions[action]}\n\n"

        case action
                when 0  :       $total += gets()
                when 1  :       $total -= gets()
                when 2  :       $total *= gets()
                when 3  :       $total /= gets()
                when 4  :       $total = 0
        end
        puts "Total = #{$total}"
end
```

Can we have that on Windows?

* You gotta be joking!!
* Why do you think we use Macs?
* How about we just turn it into a web application instead?
* Sure, we can do that with Ruby
* [What have we let ourselves in for...]

Introducing WEBrick

* WEBrick is an HTTP server library
* It's part of the Ruby 1.8 release
* It can serve static documents
* It can serve HTTPS using Ruby/OpenSSL
* It can serve arbitrary code blocks
* It can serve servlets

Static content

A standard HTTP server

```
#!/usr/local/bin/ruby
require 'webrick'

server = WEBrick::HTTPServer.new(:Port => 8080, :DocumentRoot => Dir::pwd + "/htdocs")

# mount personal directory, generating directory indexes
server.mount("/~eleanor", WEBrick::HTTPServlet::FileHandler, "/Users/eleanor/Sites", true)

# catch keyboard interrupt signal to terminate server
trap("INT"){ server.shutdown }
server.start
```

An HTTPS server

```
#!/usr/local/bin/ruby
# This requires Ruby/OpenSSL
require 'webrick'
require 'webrick/https'

certificate_name = [ ["C","UK"], ["O","games-with-brains.org"], ["CN", "WWW"] ]
server = WEBrick::HTTPServer.new(   :DocumentRoot => Dir::pwd + "/htdocs", :SSLEnable => true,
                                    :SSLVerifyClient => ::OpenSSL::SSL::VERIFY_NONE, :SSLCertName => certificate_name )
trap("INT"){ s.shutdown }
s.start
```

Servlets

A Ruby code block

```
#!/usr/local/bin/ruby
require 'webrick'

server = WEBrick::GenericServer.new()
trap("INT"){ server.shutdown }
server.start{|socket|   socket.puts("This is a code block\r")    }
```

A WEBrick servlet

```
#!/usr/local/bin/ruby
require 'webrick'

server = WEBrick::HTTPServer.new()
trap("INT"){ server.shutdown }

def generate_response(response)
      response.body = "<HTML>hello, world.</HTML>"
      response['Content-Type'] = "text/html"
end

class HelloServlet < WEBrick::HTTPServlet::AbstractServlet
      def do_GET(request, response)
            generate_response(response)
      end
end

server.mount_proc("/hello/simple"){ | request, response |  generate_response(response)   }
server.mount("/hello/advanced", HelloServlet)
server.start
```

It's that simple?

* Yes, it's that simple
* Of course these are trivial examples...
* ...so let's build an application server

An application server

* Still wondering when we get to the really good stuff?
* Soon, we promise
* But first to show you how NOT to do it!

Wrap the request

A basic request context

```
class RequestContext
        attr_reader         :request, :response, :servlets, :creation_time

        def initialize(request, response)
                @request, @response, = request, response
                @creation_time = Time.now()
        end

        def page_not_found
                @response.status = WEBrick::HTTPStatus::NotFound.new()
        end

        def response_page(page)
                @response['Content-Type'] = page.content_type
                @response.body = CGI::pretty(page.to_str())
        end

        def <<(item)
                @response.body << CGI::pretty(item)
        end
end
```

Serve the pages

The application server

```
IP_ADDRESS_PATTERN = /^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}/

class ApplicationServer
        attr_reader                             :web_server, :server_address, :servlets, :pages

        def initialize(parameters = {})
                @server_address = parameters[:my_address] or raise “Please supply a server address”
                raise “Invalid IP address for server” unless IP_ADDRESS_PATTERN.match(@server_address)
                @web_server = WEBrick::HTTPServer.new({:BindAddress => @server_address})
                @servlets = {}
                @pages = {}
        end

        def start
                trap("INT") { @web_server.shutdown }
                @web_server.start
        end

        def register_page(path, page)
                @pages[path] = page
                @web_server.mount_proc(path) { |request, response|
                        context = RequestContext.new(request, response)
                        @pages[request.path] ? context.response_page(@pages[request.path]) : context.page_not_found()
                        }
        end

        def register_method(path, handler)
                @servlets[path] = self.method(handler).to_proc
                @web_server.mount_proc(path) { |request, response|
                        context = RequestContext.new(request, response)
                        @servlets[request.path] ? (context << @servlets[request.path].call(context).to_str()) : context.page_not_found()
                        }
        end
end
```

Write the application

Revisiting "hello, world"

```
#!/usr/local/bin/ruby

require 'appserver.rb'

class SimpleServer < ApplicationServer
	def initialize(parameters = {})
		super
		register_page("/hello/simple", "<HTML>Hello, world</HTML>")
		register_method("/hello/advanced", :hello_world)
	end

	def hello_world(context)
		"<HTML>Hello, world</HTML>"
	end
end

begin
	SimpleServer.new({:my_address => ARGV.shift()}).start()

rescue RuntimeError => e
	$stderr.puts "Usage:	simpleserver	host-address"
	$stderr.puts "	address must be provided in dotted-quad format (i.e. xxx.xxx.xxx.xxx)"
end
```

What have we done?!?

* On the surface this is elegant
* But underneath it sucks
* There's no support for HTML
* Only methods can be used as servlets
* We're tied to WEBrick - which is slow

The road to perdition

* So we added an HTML 4 library
* And a server pages container
* And ActiveRecord
* We meta'd the code to death
* But it still lacked va-va-voom...

The case for Rails

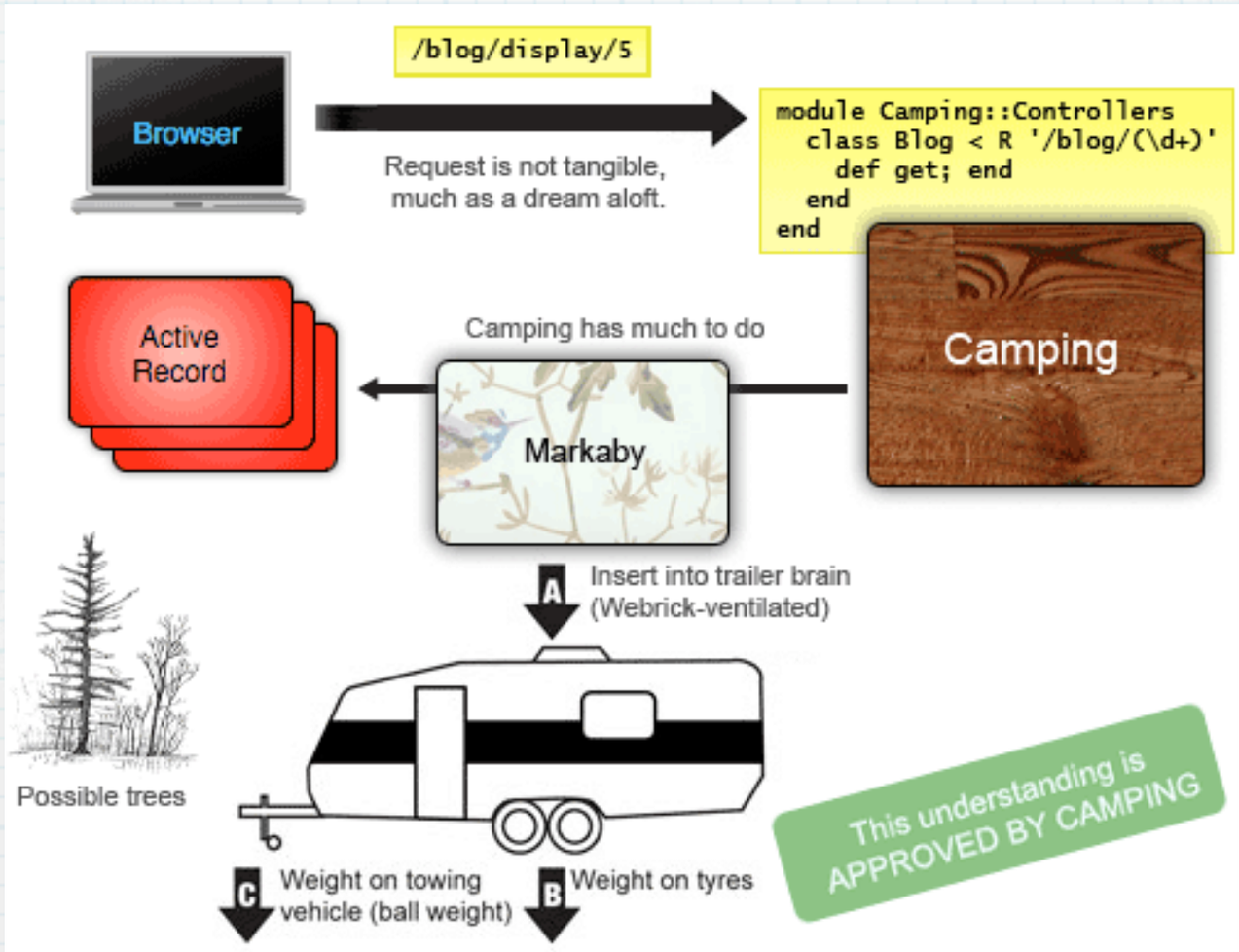
* So perhaps we should have just used Rails in the first place
* We'd be another of those "Rails saved my career" success stories!
* Hindsight's always 20/20
* But we're old-school coders and it's far too user friendly for our comfort

Are we there yet?

* Camping is beauty incarnate
* It's less than 4K of code
* It uses Markaby and ActiveRecord
* It runs on JRuby!!!
* Oh, and it's great fun to abuse...

Gratuitous diagram

How Why? The Lucky Stiff teaches it



lifted from
http://redhanded.hobix.com/bits/campingAMicroframework.html

Markaby

* An XHTML Domain Specific Language
* Allows you to embed XHTML code in Ruby code without building a complex object hierarchy
* Can be used with Rails

But that's so simple!

Markaby embedded in Ruby

```
require 'markaby'

page = Markaby::Builder.new
page.xhtml_strict do
      head { title "Camping Presentation" }
      body do
            h1.page_heading "Camping: Going off the Rails with Ruby"
            ul.page_index do
                  li.page_index { a “introduction”, :href => ‘#introduction’ }
                  li.page_index { a “the presentation”, :href => ‘/presentation’ }
                  li.page_index { a “comments”, :href => ‘#comments’ }
            end
            div.introduction! { “Everything will be alright!!!” }
            div.comments! { “Have your say” }
      end
end
puts page.to_s
```

Creates this

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-strict.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
      <head>
            <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
            <title>Camping Presentation</title>
      </head>
      <body>
            <h1 class="page_heading">Camping: Going off the Rails with Ruby</h1>
            <ul class="page_index">
                  <li class="page_index"><a href="#introduction">introduction</a></li>
                  <li class="page_index"><a href="/presentation">the presentation</a></li>
                  <li class="page_index"><a href="#comments">comments</a></li>
            </ul>

            <div id="introduction">Just breathe deeply...</div>
            <div id="comments">Have your say</div>
      </body>
</html>
```

ActiveRecord

* An Object-Relational Mapper
* Implements the Active Record pattern
* Supports many popular databases
* A key component of Rails

ORMtastic

Using Active Record

```
require 'rubygems'
require_gem ‘activerecord’

ActiveRecord::Base.establish_connection(:adapter => “sqlite3”, :host => “localhost”, :database => “test.db”)

class User < ActiveRecord::Base
end

user = User.new()
user.id = “ellie”
user.name = “Eleanor McHugh”
user.password = “somerandomtext”
user.save

user = User.find(“ellie”)
user.destroy()
```

Totally RAD

* Camping builds small applications
* Why's guideline? One file per application
* If that's how you prefer it...

A simple example

Basic setup

```
#!/usr/bin/env ruby

$:.unshift File.dirname(__FILE__) + "/../../lib"
require 'camping'
require 'camping/session'

Camping.goes :Jotter

module Blog
        include Camping::Session
end
```

* Load the camping libraries
* Define a namespace for the application
* Include session support (if required)

A simple example

Defining the data model

```
module Jotter::Models
	class Note < Base; end

	class Database < V 1.0
		def self.up
			create_table :jotter_notes, :force => true do |t|
				t.column :id,		:integer,	:null => false
				t.column :created_at,	:interger,	:null => false
				t.column :title,	:string,	:limit => 255
				t.column :body,		:text
			end
		end

		def self.down
			drop_table :jotter_notes
		end
	end
end

def Jotter.create
	Jotter::Models.create_schema
end
```

A simple example

Adding controllers

```
module Jotter::Controllers
	class Static < R '/static/(.+)'
		MIME_TYPES = {'.css' => 'text/css', '.js' => 'text/javascript', '.jpg' => 'image/jpeg'}
		PATH = __FILE__[/(.*)\//, 1]

		def get(path)
			@headers['Content-Type'] = MIME_TYPES[path[/\.\w+$/, 0]] || "text/plain"
			@headers['X-Sendfile'] = "#{PATH}/static/#{path}"
		end
	end

	class Index < R '/'
		def get
			@notes = Note.find :all
			render :index
		end
	end

	class View < R '/view/(\d+)'
		def get note_id
			@note = Note.find post_id
			render :view
		end
	end

	class Add < R ‘/add/’
		def get
			@note = Note.new
			render :add
		end

		def post
			note = Note.create :title => input.post_title, :body => input.post_body
			redirect View, post
		end
	end
```

A simple example

Adding controllers

```
class Edit < R '/edit/(\d+)', '/edit'
        def get note_id
                @note = Note.find note_id
                render :edit
        end

        def post
                @note = Note.find input.note_id
                @note.update_attributes :title => input.post_title, :body => input.post_body
                redirect View, @note
        end
end

class Delete < R '/delete/(d+)'
        def get note_id
                @note = Note.find note_id
                @note.destroy
                redirect Index
        end
end
end
```

* Respond to HTTP GET and POST requests
* Perform database operations

A simple example

Application views

```
module Jotter::Views
	def layout
		xhtml_strict do
			head do
				title 'blog'
				link :rel => 'stylesheet', :type => 'text/css', :href => '/static/styles.css', :media => 'screen'
			end

			body do
				h1.header { a 'jotter', :href => R(Index) }
				div.body do
					self << yield
				end
			end
		end
	end

	def index
		@notes.empty? (p 'No posts found.') : (ol.row! { _list_notes(@notes) })
		p { a 'new note', :href => R(Add) }
	end

	def edit
		_form(@note, :action => R(Edit))
	end

	def view
		h1 @note.title
		h2 @note.created_at
		p @note.body
		p do
			[	a("View", :href => R(View, @note)),
				a("Edit", :href => R(Edit, @note)),
				a("Delete", :href => R(View, @note))	].join " | "
		end
	end
```

A simple example

Application views

```
def _list_notes(notes)
        @notes.each do |note|
                li do
                        ul do
                                li { a note.title, :href => R(View, note) }
                                li note.created_at
                                li { a "Edit", :href => R(Edit, note) }
                                li { a "Delete", :href => R(Delete, note) }
                        end
                end
        end
end

def _form(note, opts)
        form({:method => 'post'}.merge(opts)) do
        label 'Title', :for => 'note_title'; br
        input :name => 'note_title', :type => 'text', :value => note.title; br
        label 'Body', :for => 'note_body'; br
        textarea note.body, :name => 'note_body'; br
        input :type => 'hidden', :name => 'note_id', :value => note.id
        input :type => 'submit'
end
```

* Views incorporate Markaby for XHTML
* Have access to controller data

A simple example

The post-amble

```
if __FILE__ == $0
  Jotter::Models::Base.establish_connection :adapter => 'sqlite3', :database => 'notes.db'
  Jotter::Models::Base.logger = Logger.new('camping.log')
  Jotter.create
  puts Jotter.run
end
```

A simple example

A style sheet

```
body {
        font-family: Utopia, Georga, serif;
        }

h1.header {
        background-color: #fef;
        margin: 0;
        padding: 10px;
        }

div.body {
        padding: 10px;
        }

#row ul  {
        list-style: none;
        margin: 0;
        padding: 0;
        padding-top: 4px;
        }

#row li  {
        display: inline;
        }

#row a:link, #row a:visited {
        padding: 3px 10px 2px 10px;
        color: #FFFFFF;
        background-color: #B51032;
        text-decoration: none;
        border: 1px solid #711515;
        }
```