# 1. FROM THE GROUND UP

# TABLE OF CONTENTS


01 Installing Rails

02 Creating a Rails App

03 The Command Line

04 Database Migrations

05 Ruby 1.9 Hash Syntax

06 Bundler

07 Database Configuration

# http://railsinstaller.org

Ready to deploy your app? Start a free trial at Engine Yard.

# RAILS INSTALLER GET UP & RUNNING WITH RAILS

## STEP 1

**DOWNLOAD the KIT**

WINDOWS ONLY

**For Windows only.** Mac and Linux versions *coming soon.*

*Rails Installer* has everything you need to hit the ground running. In one easy Rails Installer, you get all the common packages needed for a full Rails stack. Download it now and be writing (and running) Rails code in no time. Packages included are:

- **Ruby** 1.8.7-p334
- **Rails** 3.0.7
- **Git** 1.7.3.1
- **Sqlite** 3.7.3
- **TinyTDS** 0.4.5

## STEP 2

**WATCH the VIDEO**

In this short video, you'll see the simple installation process and how to confirm that Ruby, Rails, Git, etc are all installed properly. Watch it here or if you have trouble, view it on Vimeo.

## STEP 3

**WHAT'S NEXT?**

You've got everything installed. You've written (and ran) some code. If you find yourself asking ***where do I go from here***, don't worry. You're not alone.

The answer is ***just about anywhere***, but here's a handful of good places to start. The official Ruby on Rails site, blog, Twitter account (@rails), and mailing list. If you're into IRC, swing by #rubyonrails on irc.freenode.net. There's always the ever-growing Ruby on Rails guides. If you learn by watching, there are more screencasts

http://railstutorial.org

# Ruby on Rails Tutorial

*by Michael Hartl*

Home | Book | Help | Contact | News | Follow

skip to content | view as single page

UP

PREV NEXT

Buy Print Edition

Buy Screencasts

Rails 3 • Rails 2.3

Like 2K

Rails Tutorial Affiliate Program—50% commissions

Bonus Screencast on Haml

Ruby on Rails Tutorial

Learn Rails by Example

Michael Hartl

FOR OS X AND LINUX

## Contents


**Chapter 1 From zero to deploy**

1.1 Introduction

1.1.1 Comments for various readers

1.1.2 "Scaling" Rails

1.1.3 Conventions in this book

1.2 Up and running

# CREATING A RAILS APP

```
$ rails

Usage:
  rails new APP_PATH [options]

Options:
  -O, [--skip-active-record]     # Skip Active Record files
  -d, [--database=DATABASE]      # Preconfigure database
                                 # Default: sqlite3
  -j, [--javascript=JAVASCRIPT]  # Preconfigure JavaScript library
                                 # Default: jquery

Runtime options:
  -f, [--force]    # Overwrite files that already exist
  -p, [--pretend]  # Run but do not make any changes

Example:
    rails new ~/Code/Ruby/weblog
```

# CREATING A RAILS APP

```
$ rails new TwitterForZombies
```

```
create
create  Gemfile
create  app
create  app/controllers/application_controller.rb
create  app/mailers
create  app/models
create  app/views/layouts/application.html.erb
create  config
create  log
create  public/index.html
create  script/rails
   run  bundle install
```

Creates a bare Rails app in a new TwitterForZombies directory

# CREATING A RAILS APP

```
$ rails new TwitterForZombies
```

Installs dependencies

```
run  bundle install
Fetching source index for http://rubygems.org/
Installing rake (0.9.2)
Using activesupport (3.1)
Using rack (1.3.0)
Using actionpack (3.1)
Using actionmailer (3.1)
Using activerecord (3.1)
Using bundler (1.0.15)
Installing coffee-script (2.2.0)
Installing jquery-rails (1.0.12)
Using rails (3.1)
Installing sqlite3 (1.3.3) with native extensions
Your bundle is complete!
```

# RAILS COMMANDS

```
$ cd TwitterForZombies
$ rails
```

```
Usage: rails COMMAND [ARGS]

The most common rails commands are:
 generate    Generate new code (short-cut alias: "g")
 console     Start the Rails console (short-cut alias: "c")
 server      Start the Rails server (short-cut alias: "s")
 dbconsole   Start a console for the db in config/database.yml
                    (short-cut alias: "db")
 All commands can be run with -h for more information.
```

Lists commands

# STARTING THE SERVER

```
$ rails server
```

Starts a basic development server

```
=> Booting WEBrick
=> Rails 3.1 app in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2011-06-30 16:44:43] INFO  WEBrick 1.3.1
[2011-06-30 16:44:43] INFO  ruby 1.9.2
[2011-06-30 16:44:43] INFO  WEBrick::HTTPServer#start: port=3000
```

Shortcut

```
$ rails s
```

```
$ rails server -h
```

To lookup options

# USING GENERATORS

```
$ rails generate
```

To create template files

```
Usage: rails generate GENERATOR [args]

Please choose a generator below.

Rails:
  helper
  mailer
  migration
  model
  scaffold
```

## Shortcut

```
$ rails g
```

# STARTING WITH A SCAFFOLD

```
$ rails g scaffold
```

To create a starting point

```
Usage:
  rails generate scaffold NAME [field:type field:type]

Description:
    Scaffolds an entire resource, from model and migration to
controller and views, along with a full test suite.
```

## Example

```
$ rails g scaffold zombie name:string bio:text age:integer
```

## Variable types

| | | | | |
|---|---|---|---|---|
| string | text | integer | boolean | decimal |
| float | binary | date | time | datetime |

# STARTING WITH A SCAFFOLD

```
$ rails g scaffold zombie name:string bio:text age:integer
```

Model

```
invoke    active_record
create    db/migrate/20110701230207_create_zombies.rb
create    app/models/zombie.rb
```

Routing

```
route     resources :zombies
```

Controller

```
create    app/controllers/zombies_controller.rb
```

View

```
create    app/views/zombies
create    app/views/zombies/index.html.erb
create    app/views/zombies/edit.html.erb
create    app/views/zombies/show.html.erb
create    app/views/zombies/new.html.erb
```

PLUS TESTS, HELPERS, AND ASSETS

# INTRODUCING A MIGRATION

db/migrate/20110701230207_create_zombies.rb

Migration

```
class CreateZombies < ActiveRecord::Migration
  def change
    create_table :zombies do |t|
      t.string :name
      t.text :bio
      t.integer :age

      t.timestamps
    end
  end
end
```

AUTOMATICALLY CREATES A PRIMARY KEY CALLED ID

SAME AS

```
t.datetime :created_at
t.datetime :updated_at
```

Date & time automatically set on create and update events

# WHY WE HAVE MIGRATIONS

Story here

# RUNNING MIGRATIONS

```
db/migrate/20110701230207_create_zombies.rb
```

```
$ rails s
```

```
@zombies = Zombie.all
```

FORGOT TO RUN MIGRATIONS!

# RUNNING MIGRATIONS

```
$ rake db:migrate
```

To run all missing migrations

```
==  CreateZombies: migrating
=============================================
-- create_table(:zombies)
   -> 0.0011s
==  CreateZombies: migrated (0.0012s)
=========================================
```

```
$ rails s
```

# RAILS CONSOLE

Shortcut `$ rails c`

```
$ rails console
```

To debug our app

```
Loading development environment (Rails 3.1)
> Zombie.create(name: "Eric Allam", age: 27)
  SQL (15.2ms)  INSERT INTO "zombies" ("age", "bio", "name") VALUES
  (?, ?, ?)  [["age", 27], ["bio", nil], ["name", "Eric Allam"]]
 => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> z = Zombie.first
  Zombie Load (0.2ms)  SELECT "zombies".* FROM "zombies" LIMIT 1
 => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> z.name = "Caike Souza"
 => "Caike Souza"
```

```
> z.save
   (0.5ms)  UPDATE "zombies" SET "name" = 'Caike Souza' WHERE
   "zombies"."id" = 1
 => true
```

# RUBY 1.9 NEW HASH W/SYMBOL SYNTAX

```
Zombie.create(name: "Eric Allam", age: 27)
```

Ruby 1.9

Hash is a collection of key value pairs (key can be anything)

```
{ "first_name" => "Gregg" }
```

Ruby 1.8 & 1.9

```
{ 2 => "Pollack" }
```

Ruby 1.8 & 1.9

```
{ :name => "Eric Allam" }
```

Ruby 1.8 & 1.9

SAME AS

```
{ name: "Eric Allam" }
```

Ruby 1.9

# RUBY 1.9 NEW HASH W/SYMBOL SYNTAX

```
Zombie.create(name: "Eric Allam", age: 27)
```

Ruby 1.9

```
Zombie.create(:name => "Eric Allam", :age => 27)
```

Ruby 1.8 & 1.9

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController

  def index
    @zombies = Zombie.all

    respond_to do |format|
      format.html
      format.json { render json: @zombies }
    end
  end
  ...
```

```
render :json => @zombies
```

SAME AS

# ADDING COLUMNS

Add<Anything>To<Table name>  Column Name  Type

```
$ rails g migration AddEmailAndRottingToZombies email:string
      rotting:boolean
```

```
invoke active_record
create  db/migrate/201173223_add_email_and_rotting_to_zombies.rb
```

```
class AddEmailAndRottingToZombies < ActiveRecord::Migration
  def change
    add_column :zombies, :email, :string
    add_column :zombies, :rotting, :boolean, default: false
  end
end
```

## migration options

| | | |
|---|---|---|
| default: <value> | limit: 30 | null: false |
| first: true | after: :email | unique: true |

POSITION

# MIGRATION RAKE TASKS

```
$ rake db:migrate
```
To run all missing migrations

```
$ rake db:rollback
```
To rollback the previous migration

```
$ rake db:schema:dump
```
Dump the current db state

```
$ rake db:setup
```
Creates the db, loads schema, & seed

CALL

CREATES

When you start working on an existing app

`db/schema.rb` Authoritative source for db schema

```
ActiveRecord::Schema.define(:version => 20110703223452) do
  create_table "zombies", :force => true do |t|
    t.string   "name"
    t.text     "bio"
    t.integer  "age"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
end
```

# MIGRATION COMMANDS

Remove<Anything>From<Table name>

```
$ rails g migration RemoveAgeFromZombies age:integer
```

```
class RemoveAgeFromZombies < ActiveRecord::Migration
  def up
    remove_column :zombies, :age
  end

  def down
    add_column :zombies, :age, :integer
  end
end
```

RUN WITH ROLLBACK

## migration commands

```
rename_column :zombies, :bio, :description
rename_table :zombies, :ghouls
drop_table :zombies
change_column :zombies, :age, :integer, limit: 4
change_column_default :zombies, :is_admin, default: true
```

# MIGRATION COMMANDS

## <Anything>

```
$ rails g migration DropZombiesTable
```

```
class DropZombiesTable < ActiveRecord::Migration
  def up
    drop_table :zombies
  end

  def down
    create_table :zombies do |t|
      t.string    :name
      t.text      :bio
      t.integer   :age
      t.timestamps
    end
  end
end
```

# http://guides.rubyonrails.org/

More at rubyonrails.org: Overview | Download | Deploy | Code | Screencasts | Documentation | Ecosystem | Community | Blog

RailsGuides

Home | Guides Index | Contribute | Credits

## Ruby on Rails Guides (v3.0.9)

These are the new guides for Rails 3. The guides for Rails 2.3 are still available at http://guides.rubyonrails.org/v2.3.8/.

These guides are designed to make you immediately productive with Rails, and to help you understand how all of the pieces fit together.

Rails Guides are a result of the ongoing Guides hackfest, and a work in progress.

Guides marked with this icon are currently being worked on. While they might still be useful to you, they may contain incomplete information and even errors. You can help by reviewing them and posting your comments and corrections to the author.

### Start Here

**Getting Started with Rails**

Everything you need to know to install Rails and create your first application.

### Models

**Rails Database Migrations**

This guide covers how you can use Active Record migrations to alter your database in a structured and organized manner.

# BUNDLER

/Gemfile

Contains all your application dependencies

```
source 'http://rubygems.org'

gem 'rails', '3.1'
gem 'sqlite3'

# Asset template engines
gem 'sass-rails', "~> 3.1"
gem 'coffee-script'
gem 'uglifier'

gem 'jquery-rails'
```

```
$ bundle install
```

Installs all application dependencies

Initially called by ‘rails new’

# http://rubyonrails.org/screencasts/rails3

Overview | Download | Deploy | Bugs/Patches | Screencasts | Documentation | Ecosystem | Community | Blog

## Dive into Rails 3

Learn about all the new components like Bundler, Active Model, and Active Relation, as well as the new syntax of the router and Action Mailer. It's everything you need to get started.

### Bundler & Action Mailer

Download
Quicktime HD
iPhone/iPod

Subscribe
Video RSS

**Description**

Rails 3 ships with Bundler, a Ruby library that makes Dependency Management a painless process. In this video we learn Bundler basics and take a look at the new

# http://gembundler.com

0.9 **v1.0**

Fork me on GitHub

The best way to manage your application's dependencies

# Bundler

Bundler manages an **application's dependencies** through its entire life across many machines systematically and repeatably.

## I am interested in

Troubleshooting | Understanding Bundler | Gemfile Manual | CLI Manual

## Getting Started

Getting started with bundler is easy

```
$ gem install bundler
```

Specify your dependencies in a Gemfile in your project's root

# CONFIGURING YOUR DATABASE

From Gemfile

```
gem 'sqlite3'
```

/config/database.yml

Contains your database configuration

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000

test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000
```

For Mysql

```
database:
  adapter: mysql2
  encoding: utf8
  database: tfzombies
  pool: 5
  username: root
  password: secret
  socket: /tmp/mysql.sock
```

In Gemfile

```
gem 'mysql2'
```

http://bit.ly/rfz2beta

to get the course....

Do Level 1

# 2. MODELS STILL TASTE LIKE CHICKEN

# TABLE OF CONTENTS


01 Named Scope

02 Callbacks

03 has_one

04 Relationship Options

05 Using Includes

06 has_many :through

# GOOD CODE AND BAD CODE

BAD CODE

GOOD CODE

NOTHING IN PARTICULAR

BRAINS - NO SUGAR ADDED

# NAMED SCOPE

app/controllers/rotting_zombies_controller.rb

```
class RottingZombiesController < ApplicationController

  def index
    @rotting_zombies = Zombie.where(rotting: true)
    ...
  end

end
```

WE'RE OFTEN GOING TO QUERY

FOR ROTTING ZOMBIES

# NAMED SCOPE

app/controllers/rotting_zombies_controller.rb

```
class RottingZombiesController < ApplicationController

  def index
    @rotting_zombies = Zombie.rotting
    ...
  end

end
```

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  scope :rotting, where(rotting: true)

end
```

# NAMED SCOPE

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  scope :rotting, where(rotting: true)
  scope :fresh, where("age < 20")
  scope :recent, order("created_at desc").limit(3)

end
```

Use method chaining to create queries

```
Zombie.rotting.limit(5)
```

```
Zombie.recent.rotting
```

```
Zombie.recent.fresh.rotting
```

# CALLBACKS

**Objective**: When zombie age > 20, set **rotting** to true

app/controllers/zombies_controller.rb

```
def update
  @zombie = Zombie.find(params[:id])

  if @zombie.age > 20
    @zombie.rotting = true
  end

  respond_to do |format|
    if @zombie.update_attributes(params[:zombie])
      ...
    else
      ...
    end
  end
end
```

# CALLBACKS

**Objective**: When zombie age > 20, set **rotting** to true

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  before_save :make_rotting

  def make_rotting
    if @zombie.age > 20
      @zombie.rotting = true
    end
  end

end
```

INVALID VARIABLES

# CALLBACKS

**Objective**: When zombie age > 20, set **rotting** to true

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  before_save :make_rotting

  def make_rotting
    if age > 20
      self.rotting = true
    end
  end

end
```

```
if @zombie.age > 20
  @zombie.rotting = true
end
```

reading attributes doesn't need self

setting attributes needs "self."

# CALLBACKS

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  before_save :make_rotting

  def make_rotting
    if age > 20
      self.rotting = true
    end
  end

end
```

REFACTOR

```
def make_rotting
  self.rotting = true if age > 20
end
```

# RETURNING FALSE WILL HALT

app/models/brain.rb

```
class Brain < ActiveRecord::Base
  before_save :taste

  def taste
    return false
  end
```

SAME AS

```
def taste
  false
end
```

save **stops** if callback returns false

```
> b = Brain.new(:flavor => 'butter')
> b.save
 => false
```

BE CAREFUL OF RETURN VALUES

# ALL CALLBACKS

```
before_validation
after_validation
```

```
before_save
after_save
```

```
before_create
after_create
```

```
before_update
after_update
```

```
before_destroy
after_destroy
```

1. Can use multiple callbacks
   (even of same type)
2. If any return false then everything stops
   (with before then save/destroy isn't done)

Examples

```
after_create :send_welcome_email
```

```
before_save :encrypt_password
```

```
before_destroy :set_deleted_flag
```

# TABLE OF CONTENTS


01 Named Scope

02 Callbacks

**03 has_one**

04 Relationship Options

05 Using Includes

06 has_many :through

A Tweet **belongs to** a Zombie

app/models/tweet.rb

```
class Tweet < ActiveRecord::Base

  belongs_to :zombie

end
```

Singular

A Zombie **has many** Tweets

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base

  has_many :tweets

end
```

Plural

**tweets**

| id | status | zombie_id |
|---|---|---|
| 1 | Where can I get a good bite to eat? | 1 |
| 2 | My left arm is missing, but I don't care. | 2 |
| 3 | I just ate some delicious brains. | 3 |
| 4 | OMG, my fingers turned green. #FML | 1 |

**zombies**

| id | name | graveyard |
|---|---|---|
| 1 | Ash | Glen Haven Memorial Cemetery |
| 2 | Bob | Chapel Hill Cemetery |
| 3 | Jim | My Father's Basement |

# HAS___ONE

**Objective:** A zombie may or may not have a (single) brain

# HAS_ONE

**Objective**: A zombie may or may not have a (single) brain

```
$ rails g model brain zombie_id:integer status:string flavor:string
```

```
invoke    active_record
create    db/migrate/20110805011138_create_brains.rb
create    app/models/brain.rb
```

Model

db/migrate/20110805011138_create_brains.rb

```
class CreateBrains < ActiveRecord::Migration
  def change
    create_table :brains do |t|
      t.integer :zombie_id
      t.string :status
      t.string :flavor
    end
    add_index :brains, :zombie_id
  end
end
```

add a foreign_key index

```
$ rake db:migrate
```

# HAS_ONE

**Objective**: A zombie may or may not have a (single) brain

app/models/brain.rb

```ruby
class Brain < ActiveRecord::Base
  belongs_to :zombie
end
```

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  has_one :brain
end
```

```
$ rails console
Loading development environment (Rails 3.1.0)
> z = Zombie.last
    => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> z.create_brain(status: "Squashed", flavor: "Mud")
    => #<Brain id: 1, zombie_id: 1, status: "Squashed", flavor: "Mud">
```

```
> z.brain
    => #<Brain id: 1, zombie_id: 1, status: "Squashed", flavor: "Mud">
```

# WHAT HAPPENS WHEN WE DESTROY ZOMBIE?

```
$ rails console
```

```
Loading development environment (Rails 3.1.0)
> z = Zombie.find(1)
  => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> Brain.find(1)
  => #<Brain id: 1, zombie_id: 1, status: "Squashed", flavor: "Mud">
```

```
> z.destroy
  SQL (0.4ms)  DELETE FROM "zombies" WHERE "zombies"."id" = 1
  => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> Brain.find(1)
  => #<Brain id: 1, zombie_id: 1, status: "Squashed", flavo
```

ARG! ZOMBIE BRAIN STILL LIVES!

has_one

**has_one**(association_id, options = {})
ActiveRecord::Associations::ClassMethods
Specifies a one-to-one association with another class. Th

**has_one**_association.rb
files/activerecord/lib/active_record/associations/has_or

**has_one**_through_association.rb
files/activerecord/lib/active_record/associations/has_or

expand_hash_conditions_for_aggregates(attrs)
ActiveRecord::Base
Accepts a hash of SQ

has_flash_with_c
ActionDispatch::TestR
Do we have a flash th

has_session_obje
ActionDispatch::TestR
Does the specified ob

new_from_hash_c
ActiveSupport::HashWithIndifferentAccess

sanitize_sql_hash_for_assignment(attrs)
ActiveRecord::Base
Sanitizes a hash of attribute/value pairs into SQL conditio

## **has_one**(association_id, options = {})

Specifies a one-to-one association with another class. This method should only be used if the other class contains the f
If the current class contains the foreign key, then you should use `belongs_to` instead. See also ActiveRecord::Associations::ClassMethods's overview on when to use has_one and when to use belongs_to.

The following methods for retrieval and query of a single associated object will be added:

**association(force_reload = false)**
Returns the associated object. `nil` is returned if none is found.

**association=(associate)**
associate object.

ked to this object th
s. It will NOT work i

to this object throu

(`association` is replaced with the symbol passed as the first argument, so `has_one :manager` would add among other
`manager.nil?`.)

### Example

An Account class declares `has_one :beneficiary`, which will add:

- `Account#beneficiary` (similar to `Beneficiary.find(:first, :conditions => "account_id = #{id}")`)
- `Account#beneficiary=(beneficiary)` (similar to `beneficiary.account_id = account.id; beneficiary.save`)
- `Account#build_beneficiary` (similar to `Beneficiary.new("account_id" => id)`)
- `Account#create_beneficiary` (similar to `b = Beneficiary.new("account_id" => id); b.save; b`)

### Options

The declaration can also include an options hash to specialize the behavior of the association.

Options are:

**:class_name**
Specify the class name of the association. Use it only if that name can't be inferred from the association name. So
`:manager` will by default be linked to the Manager class, but if the real class name is Person, you'll have to specify
ion.

**:conditions**

**:dependent**

If set to `:destroy`, the associated object is destroyed when this object is. If set to `:delete`, the associated object is deleted **without** calling its destroy method. If set to `:nullify`, the associated object's foreign key is set to `NULL`. Also, association is assigned.

http://railsapi.com

# DESTROYING THE ZOMBIE'S BRAIN

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  has_one :brain, dependent: :destroy
end
```

```
$ rails console
```

```
Loading development environment (Rails 3.1.0)
> z = Zombie.find(1)
    => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> z.destroy
  SQL (0.5ms)  DELETE FROM "brains" WHERE "brains"."id" = 1
  SQL (0.2ms)  DELETE FROM "zombies" WHERE "zombies"."id" = 1
```

# RELATIONSHIP OPTIONS

| | |
|---|---|
| `dependent: :destroy` | will call destroy on associated objects |
| `foreign_key: :undead_id` | changes the associated key (i.e. zombie_id) |
| `primary_key: :zid` | changes the primary key (i.e. id) |
| `validate: true` | when zombie validates brain will too |

And many more

(read the documentation)

# RELATIONSHIP "INCLUDE" OPTION

app/controllers/zombies_controller.rb

```
def index
  @zombies = Zombie.all
```

app/views/zombies/index.html.erb

```
<% @zombies.each do |zombie| %>
<tr>
  <td><%= zombie.name %></td>
  <td><%= zombie.brain.flavor %></td>
</tr>
<% end %>
```

## Listing zombies

| Name | Brain Flavor |
|---|---|
| Joe | Mud |
| Jim | Strawberry |
| Bob | Butter |
| Tony | Bubble Gum |

QUERY FOR EACH BRAIN!
N + 1 ISSUE

Server runs

```
Zombie Load (0.1ms)  SELECT * FROM "zombies"
  Brain Load (0.2ms)  SELECT * FROM "brains" WHERE "zombie_id" = 4
  Brain Load (0.1ms)  SELECT * FROM "brains" WHERE "zombie_id" = 5
  Brain Load (0.1ms)  SELECT * FROM "brains" WHERE "zombie_id" = 6
  Brain Load (0.1ms)  SELECT * FROM "brains" WHERE "zombie_id" = 7
```

# RELATIONSHIP "INCLUDE" OPTION

app/controllers/zombies_controller.rb

```
def index
  @zombies = Zombie.includes(:brain).all
```

app/views/zombies/index.html.erb

```
<% @zombies.each do |zombie| %>
<tr>
  <td><%= zombie.name %></td>
  <td><%= zombie.brain.flavor %></td>
</tr>
<% end %>
```

## Listing zombies

| Name | Brain Flavor |
|---|---|
| Joe | Mud |
| Jim | Strawberry |
| Bob | Butter |
| Tony | Bubble Gum |

## In the log files

```
Zombie Load (1.8ms) SELECT * FROM "zombies"
Brain Load (0.3ms) SELECT * FROM "brains" WHERE "zombie_id" IN (4, 5, 6, 7)
```

# HAS_MANY :THROUGH

**Objective**: A zombie may play multiple roles

**Captain**

**Scout**

**Soldier**

**Brain Taster**

# HAS_MANY :THROUGH

**Objective:** A zombie may play multiple roles

role_id

scout
captain
soldier
brain taster

ONLY GOING TO WORK IF A ZOMBIE HAS ONE ROLE!

# HAS_MANY :THROUGH

**Objective:** A zombie may play multiple roles

# HAS_MANY :THROUGH

**Objective**: A zombie may play multiple roles

db/migrate/xxx_create_assignments.rb

```ruby
class CreateAssignments < ...
  def change
    create_table :assignments do |t|
      t.integer :zombie_id
      t.integer :role_id
    end
    add_index :assignments, :zombie_id
    add_index :assignments, :role_id
  end
end
```

db/migrate/xxx_create_roles.rb

```ruby
class CreateRoles < ...
  def change
    create_table :roles do |t|
      t.string :title
    end
  end
end
```

# has_many :through

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  has_many :assignments
  has_many :roles, through: :assignments
end
```

app/models/assignment.rb

```ruby
class Assignment < ...
  belongs_to :zombie
  belongs_to :role
end
```

app/models/role.rb

```ruby
class Role < ActiveRecord::Base
  has_many :assignments
  has_many :zombies, through: :assignments
end
```

# HAS_MANY :THROUGH

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  has_many :assignments
  has_many :roles, through: :assignments
end
```

```
$ rails console
```

```
> z = Zombie.last
   => #<Zombie id: 1, name: "Eric Allam", bio: nil, age: 27>
```

```
> z.roles << Role.find_by_title("Captain")
Role Load (0.2ms) SELECT * FROM "roles" WHERE "title" = 'Captain' LIMIT 1
SQL (0.4ms) INSERT INTO "assignments" ("role_id", "zombie_id") VALUES (2, 1)
```

<< is a function on Array  it's the same as z.roles.push

```
> z.roles
Role Load (0.3ms)  SELECT "roles".* FROM "roles" INNER JOIN "assignments" ON
"roles"."id" = "assignments"."role_id" WHERE "assignments"."zombie_id" = 1
```

# 3. REST IN PIECES

# TABLE OF CONTENTS


01 Understanding REST

02 Revisit URL Helpers

03 Forms & Input Helpers

04 Nested Resources

05 View Partials

06 Other View Helpers

# WHAT IS REST? Representational State Transfer

**Roy Fielding**

published a dissertation in 2000 entitled

*"Architectural Styles and the Design of Network-based Software Architectures"*

Example of REST

*HTTP Protocol*

1. Resources (noun)
   addressable through URI

2. Standard methods (verb)
   GET, POST, PUT, DELETE

RESTful Web Services
by Leonard Richardson
& Sam Ruby

FOR MORE INFO

# HOW DID RAILS BECOME RESTFUL?

app/controllers/users_controller.rb

```
class UserController < ApplicationController

  def login_to_website
    ...
  end

  def subscribe_to_mailing_list
    ...
  end

  def process_credit_card_transaction
    ...
  end

end
```

Bad Because...

1. Lends itself to huge controllers
2. Multiple models per controller

# RAILS 1 RECOMMENDED REST

app/controllers/users_controller.rb

```
class UserController < ApplicationController

  # GET /users/show/3
  def show

  # POST /users/create
  def create

  # POST /users/update/3
  def update

  # POST /users/destroy/3
  def destroy

end
```

Bad Because...

We're repeating the VERB!

# USING PROPER REST

**Rails 1**

| Actions | show | create | update | destroy |
|---|---|---|---|---|
| SQL | select | create | update | delete |
| REST | get | post | post | post |

**Rails 2+**

| Actions | show | create | update | destroy |
|---|---|---|---|---|
| SQL | select | create | update | delete |
| REST | get | post | **put** | **delete** |

USING THE REST VERBS

# USING PROPER REST

| Verb | Noun |
|---|---|
| GET | /users/show/3 |
| POST | /users/create |
| POST | /users/update/3 |
| POST | /users/destroy/3 |

**Rails 1**

| Verb | Noun |
|---|---|
| GET | /users/3 |
| POST | /users |
| PUT | /users/3 |
| DELETE | /users/3 |

**Rails 2+**

WITH PROPER VERBS WE CAN SIMPLIFY THE URI

# REST CHALLENGES

Activity

**Show user**

||

Verb

**?**

Noun

**?**

# REST CHALLENGES

Activity

**Show user**

||

Verb

**GET**

Noun

**User**

/users/3

# REST CHALLENGES

Activity

**Create user**

||

Verb

**POST**

Noun

**User**

/users

# REST CHALLENGES

Activity

**Update user**

||

Verb

**PUT**

Noun

**User**

/users/3

# REST CHALLENGES

Activity

**Destroy user**

||

Verb

**DELETE**

Noun

**User**

/users/3

Activity

**Play Global Thermonuclear War**

||

Verb

**POST**

Noun

**GlobalThermonuclearWar**

`/global_thermonuclear_wars`

# REST CHALLENGES

Activity

**Process credit card transaction**

||

Verb

**POST**

Noun

**Transaction**

`/orders/5/credit_card_transactions`

# REST CHALLENGES

Activity

**Subscribe to mailing list**

||

Verb

**POST**

Noun

**Subscription**

`/mailing_lists/5/subscriptions`

# REST CHALLENGES

Activity

**Logout**

||

Verb

**DELETE**

Noun

**Session**

`/sessions`

# REVISED RAILS CONTROLLERS

```ruby
class UserController

  # GET /users/show/3
  def show

  # POST /users/create
  def create

  # POST /users/update/3
  def update

  # POST /users/destroy/3
  def destroy

end
```

**Rails 1**

```ruby
class UserController

  # GET /users/3
  def show

  # POST /users
  def create

  # PUT /users/3
  def update

  # DELETE /users/3
  def destroy

end
```

**Rails 2+**

# RESTFUL LINKS

## To link to each action

```
class UserController

  # GET /users/3
  def show

  # POST /users
  def create

  # PUT /users/3
  def update

  # DELETE /users/3
  def destroy

end
```

```
zombie = Zombie.find(2)
```

```
<%= link_to 'show', zombie %>
```

```
<%= link_to 'create', zombie, method: :post %>
```

```
<%= link_to 'update', zombie, method: :put %>
```

```
<%= link_to 'delete', zombie, method: :delete %>
```

BUT BROWSERS DON'T SUPPORT PUT AND DELETE?!?!

# EXPLAINING PUT AND DELETE

```
<%= link_to 'update', zombie, method: :put %>
```

```
<%= link_to 'delete', zombie, method: :delete %>
```

```
<a href="/zombies/4" data-method="put" rel="nofollow">update</a>
<a href="/zombies/4" data-method="delete" rel="nofollow">delete</a>
```

HTML5 data attribute

Creates by unobtrusive JavaScript

```
<form method="post" action="/zombies/4">
  <input name="_method" value="delete" />
</form>
```

_method set as method (verb) by Rails

(All under the covers)

# RAKE ROUTES

app/config/routes.rb

```
resources :zombies
```

Creates our RESTful routes

```
$ rake routes
```

```
     zombies GET    /zombies           {:action=>"index", :controller=>"zombies"}
             POST   /zombies           {:action=>"create", :controller=>"zombies"}
 new_zombie GET    /zombies/new       {:action=>"new", :controller=>"zombies"}
edit_zombie GET    /zombies/:id/edit  {:action=>"edit", :controller=>"zombies"}
      zombie GET    /zombies/:id       {:action=>"show", :controller=>"zombies"}
             PUT    /zombies/:id       {:action=>"update", :controller=>"zombies"}
             DELETE /zombies/:id       {:action=>"destroy", :controller=>"zombies"}
```

```
<%= link_to 'All Zombies', zombies_path %>
```

```
<%= link_to 'New Zombie',  new_zombie_path %>
```

```
<%= link_to 'Edit Zombie', edit_zombie_path(@zombie) %>
```

```
<%= link_to 'Show Zombie', zombie_path(@zombie) %>
```

```
<%= link_to 'Show Zombie', @zombie %>
```

# PATH AND URL HELPERS

## Relative Path

| | |
|---|---|
| zombies_path | /zombies |
| new_zombie_path | /zombies/new |

## Absolute Path

| | |
|---|---|
| zombies_url | http://localhost:3000/zombies |
| new_zombie_url | http://localhost:3000/zombies/new |

# TABLE OF CONTENTS


01 Understanding REST

02 Revisit URL Helpers

03 Forms & Input Helpers

04 Nested Resources

05 View Partials

06 Other View Helpers

# CREATING A FORM

Both create and update zombie form

```
<%= form_for(@zombie) do |f| %>
  ...
<% end %>
```

If `@zombie` isn't saved to the database yet

```
<form action="/zombies" method="post">
```

If `@zombie` is saved to the database

```
<form action="/zombies/8" method="post">
  <input name="_method" type="hidden" value="put" />
```

# SUBMIT BUTTON

Both create and update zombie form

```
<%= form_for(@zombie) do |f| %>
  ...
  <%= f.submit %>
<% end %>
```

If `@zombie` isn't saved to the database yet

```
<input name="commit" type="submit" value="Create Zombie" />
```

If `@zombie` is already saved to the database

```
<input name="commit" type="submit" value="Update Zombie" />
```

# TEXT_FIELD HELPER

```
<%= form_for(@zombie) do |f| %>
  ...
    <%= f.text_field :name %>
  <%= f.submit %>
<% end %>
```

When submitted, request parameters

```
:params => {:zombie => {:name => "Eric"}}
```

Can be viewed from log

If `@zombie` isn't saved to the database yet

```
<input name="zombie[name]" size="30" type="text" />
```

If `@zombie` is already saved to the database

```
<input name="zombie[name]" size="30" type="text" value="Eric" />
```

If `@zombie.name` has a validation error

```
<div class="field_with_errors">
  <input name="zombie[name]" size="30" type="text" value="" />
</div>
```

# LABEL HELPER

```
<%= form_for(@zombie) do |f| %>
  ...
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  <%= f.submit %>
<% end %>
```

Will render out

```
<label for="zombie_name">Name</label>
```

If `@zombie.name` has a validation error

```
<div class="field_with_errors">
  <label for="zombie_name">Name</label>
</div>
```

# INPUT HELPERS

```
<%= f.text_area :bio %>
```

Renders a multiline text area

```
<%= f.check_box :rotting %>
```

Check box used for booleans

List of radio buttons (without their labels)

```
<%= f.radio_button :decomp, 'fresh', checked: true %>
<%= f.radio_button :decomp, 'rotting' %>
<%= f.radio_button :decomp, 'stale' %>
```

Select box with three options

```
<%= f.select :decomp, ['fresh', 'rotting', 'stale'] %>
```

Select box with three options, each with a numerical value

```
<%= f.select :decomp, [['fresh', 1], ['rotting', 2], ['stale', 3]] %>
```

# ALTERNATE TEXT INPUT HELPERS

type="<field>"

```
<%= f.password_field :password %>
```

```
<%= f.number_field :price %>
```

```
<%= f.range_field :quantity %>
```

```
<%= f.email_field :email %>
```

```
<%= f.url_field :website %>
```

```
<%= f.telephone_field :mobile %>
```

# NESTED ROUTES

app/config/routes.rb

```
TwitterForZombies::Application.routes.draw do

  resources :zombies
  resources :tweets

end
```

/tweets/2 To find a tweet

/tweets?zombie_id=4 To find all zombie's tweet

NOT VERY RESTFUL,

TWEETS SHOULDN'T EXIST WITHOUT A ZOMBIE!!

# NESTED ROUTES

app/config/routes.rb

```
TwitterForZombies::Application.routes.draw do

  resources :zombies do
    resources :tweets
  end

end
```

/zombies/4/tweets/2 — To find a tweet

/zombies/4/tweets — To find all zombie's tweet

To make this work, we must:

1. Update the controller
2. Update all the links & form_for

# UPDATING THE CONTROLLER FOR NESTING

/zombies/4/tweets/2

params = {:zombie_id => 4, :id => 2}

/zombies/4/tweets

params = {:zombie_id => 4}

app/controller/tweets_controller.rb

```
class TweetsController < ApplicationController
  before_filter :get_zombie

  def get_zombie
    @zombie = Zombie.find(params[:zombie_id])
  end

  def show
    @tweet = @zombie.tweets.find(params[:id])
    ...
  end

 end
```

# UPDATING THE CONTROLLER FOR NESTING

/zombies/4/tweets

```
params = {:zombie_id => 4}
```

app/controller/tweets_controller.rb

```
class TweetsController < ApplicationController
  before_filter :get_zombie

  def get_zombie
    @zombie = Zombie.find(params[:zombie_id])
  end

  def index
    @tweets = @zombie.tweets
    ...
  end

end
```

# NESTED RAKE ROUTES

```
$ rake routes
```

```
    zombie_tweets GET    /zombies/:zombie_id/tweets
                  POST   /zombies/:zombie_id/tweets
 new_zombie_tweet GET    /zombies/:zombie_id/tweets
edit_zombie_tweet GET    /zombies/:zombie_id/tweets/:id/edit
     zombie_tweet GET    /zombies/:zombie_id/tweets/:id
                  PUT    /zombies/:zombie_id/tweets/:id
                  DELETE /zombies/:zombie_id/tweets/:id
```

```
<%= link_to "#{@zombie.name}'s Tweets", zombie_tweets_path(@zombie) %>
```

```
<%= link_to 'New Tweet', new_zombie_tweet_path(@zombie) %>
```

```
<%= link_to 'Edit', edit_zombie_tweet_path(@zombie, tweet) %>
```

```
<%= link_to 'Show', zombie_path(@zombie, tweet) %>
```

```
<%= link_to 'Show', [@zombie, tweet] %>
```

```
<%= link_to 'Destroy', [@zombie, tweet], method: :delete %>
```

# UPDATING THE VIEW FOR NESTING

app/views/tweets/index.html.erb

```
<% @tweets.each do |tweet| %>
 <tr>
  <td><%= tweet.body %></td>
  <td><%= link_to 'Show', [@zombie, tweet] %></td>
  <td><%= link_to 'Edit', edit_zombie_tweet_path(@zombie, tweet) %></td>
  <td><%= link_to 'Destroy', [@zombie, tweet], method: :delete %></td>
 </tr>
<% end %>

<%= link_to 'New Tweet', new_zombie_tweet_path(@zombie) %>
```

app/views/tweets/_form.html.erb

```
<%= form_for([@zombie, @tweet]) do |f| %>
```

# UPDATING ROUTES IN THE CONTROLLER

app/controller/tweets_controller.rb

```
...
def create
  @tweet = Tweets.new(params[:tweet])

  respond_to do |format|
    if @tweet.save
      format.html { redirect_to @tweet,
                    notice: 'Tweet was successfully created.' }
      format.json { render json: @tweet,
                           status: :created,
                           location: @tweet }
    else
      format.html { render action: "new" }
      format.json { render json: @tweet.errors,
                                status: :unprocessable_entity }
    end
  end
end
```

# UPDATING ROUTES IN THE CONTROLLER

app/controller/tweets_controller.rb

```
...
def create
  @tweet = @zombie.tweets.new(params[:tweet])

  respond_to do |format|
    if @tweet.save
      format.html { redirect_to [@zombie, @tweet],
                    notice: 'Tweet was successfully created.' }
      format.json { render json: [@zombie, @tweet],
                           status: :created,
                           location: [@zombie, @tweet] }
    else
      format.html { render action: "new" }
      format.json { render json: @tweet.errors,
                                status: :unprocessable_entity }
    end
  end
end
```

# TABLE OF CONTENTS


01 Understanding REST

02 Revisit URL Helpers

03 Forms & Input Helpers

04 Nested Resources

05 View Partials

06 Other View Helpers

# PARTIALS - HOW WE REUSE VIEW CODE

Partials start with an underscore

app/views/tweets/_form.html.erb

```
<%= form_for([@zombie, @tweet]) do |f| %>
```

app/views/tweets/new.html.erb

```
<h1>New tweet</h1>

<%= render 'form' %>

<%= link_to 'Back', zombie_tweets_path(@zombie) %>
```

app/views/tweets/edit.html.erb

```
<h1>Editing tweet</h1>

<%= render 'form' %>
```

# VIEW HELPERS – MAKING STRINGS SAFE

**Rails 2**

```
<%= @tweet.body %>
```

(unsafe)

**Rails 2**

```
<%= h @tweet.body %>
```

(safe)

**Rails 3**

```
<%= raw @tweet.body %>
```

(unsafe)

**Rails 3**

```
<%= @tweet.body %>
```

(safe)

# ADDITIONAL VIEW HELPERS

```
<% @tweets.each do |tweet| %>
  <div id="tweet_<%= tweet.id %>" class="tweet">
    <%= tweet.body %>
  </div>
<% end %>
```

SAME AS

```
<% @tweets.each do |tweet| %>
  <%= div_for tweet do %>
    <%= tweet.body %>
  <% end %>
<% end %>
```

```
dom_id(@tweet)
```

→ `tweet_2`

# ADDITIONAL VIEW HELPERS

```
<%= truncate("I need brains!", :length => 10) %>
```

I need bra...

```
<%= truncate("I need brains!", :length => 10, :separator => ' ') %>
```

I need...

```
I see <%= pluralize(Zombie.count, "zombie") %>
```

with 1 I see 1 zombie

with 2 I see 2 zombies

```
His name was <%= @zombie.name.titleize %>
```

His name was Ash Williams

# ADDITIONAL VIEW HELPERS

```
Ash's zombie roles are <%= @role_names.to_sentence %>
```

Ash's zombie roles are Captain, Soldier, and Brain Taster

```
He was buried alive <%= time_ago_in_words @zombie.created_at %> ago
```

He was buried alive 2 days ago

```
Price is <%= number_to_currency 13.5 %>
```

Price is $13.50

```
Ash is <%= number_to_human 13234355423 %> years old
```

Ash is 13.2 billion years old

http://guides.rubyonrails.org/



RailsGuides



# Active Support Core Extensions

Active Support is the Ruby on Rails component responsible for providing Ruby language extensions, utilities, and other transversal stuff.

It offers a richer bottom-line at the language level, targeted both at the development of Rails applications, and at the development of Ruby on Rails itself.

By referring to this guide you will learn the extensions to the Ruby core classes and modules provided by Active Support.

## Chapters

1. How to Load Core Extensions
   - Stand-Alone Active Support
   - Active Support Within a Ruby on Rails Application
2. Extensions to All Objects
   - blank? and present?
   - presence
   - duplicable?
   - try
   - singleton_class
   - class_eval(*args, &block)

## 1 How to Load Core Extensions

### 1.1 Stand-Alone Active Support

In order to have a near zero default footprint, Active Support does not load anything by default. It is broken in small pieces so that you may load just what you need, and also has some convenience entry points to load related extensions in one shot, even everything.

# 4. ASSET PACKAGING AND MAILING

# TABLE OF CONTENTS


01 Creating and Sending Mail

02 Sending Attachments in Mail

03 The Asset Pipeline

04 Asset Tags

05 CoffeeScript

06 SCSS

07 JavaScript Manifest

# CREATE A ZOMBIE MAILER

**Objective**: Create emails for decomposition change and lost brain

```
$ rails g mailer ZombieMailer decomp_change lost_brain
```

```
    create    app/mailers/zombie_mailer.rb
    invoke    erb
    create    app/views/zombie_mailer
    create    app/views/zombie_mailer/decomp_change.text.erb
    create    app/views/zombie_mailer/lost_brain.text.erb
```

Mailer

app/mailers/zombie_mailer.rb

```
class ZombieMailer < ActionMailer::Base
  default from: "from@example.com"

  def decomp_change
    @greeting = "Hi"

    mail to: "to@example.org"
  end
  ...
end
```

INSTANCE VARS WE WANT IN OUR VIEWS

# CREATE A ZOMBIE MAILER

app/mailers/zombie_mailer.rb

```ruby
class ZombieMailer < ActionMailer::Base
  default from: "from@example.com"

  def decomp_change(zombie)
    @zombie = zombie
    @last_tweet = @zombie.tweets.last

    attachments['z.pdf'] = File.read("#{Rails.root}/public/zombie.pdf")
    mail to: @zombie.email, subject: 'Your decomp stage has changed'
  end
  ...
end
```

```ruby
from: my@email.com
cc: my@email.com
bcc: my@email.com
reply_to: my@email.com
```

CAN ALSO BE DEFAULTS

# MAILER VIEWS

app/views/zombie_mailer/decomp_change.text.erb

```
Greetings <%= @zombie.name %>,

  Your decomposition state is now <%= @zombie.decomp %> and your
last tweet was: <%= @last_tweet.body %>

Good luck!
```

If you want HTML emails, you just rename text to html

app/views/zombie_mailer/decomp_change.html.erb

```
<h1>Greetings <%= @zombie.name %>,</h1>

<p>Your decomposition state is now <%= @zombie.decomp %> and your
last tweet was: <%= @last_tweet.body %></p>

<%= link_to "View yourself", zombie_url(@zombie) %>
```

If you want multipart (both html & text) just make 2 files

# SENDING MAIL

app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  after_save :decomp_change_notification, if: :decomp_changed?

  private

  def decomp_change_notification
    ZombieMailer.decomp_change(self).deliver
  end
end
```

# MAD MIMI

Mass mailing is best done outside of Rails.

Gemfile `gem 'madmimi'` then 'bundle install'

```
mimi = MadMimi.new('<email>', '<api_key>')
```

madmimi.com

```
mimi.add_to_list('gregg@envylabs.com', 'newsletter')
```

```
mimi.remove_from_list('gregg@envylabs.com', 'newsletter')
```

```
mimi.memberships('gregg@envylabs.com')
```

**Monthly Newsletter** ☑

Our newsletter contains information about new courses, live events, and other Code School news

The Code School Newsletter – August 2011 — All Mail

Delete Junk Reply Reply All Forward Print To Do

From: Code School <newsletter@codeschool.com>
Subject: **The Code School Newsletter - August 2011**
Date: August 10, 2011 12:05:41 PM EDT
To: gregg+test@envylabs.com

code school
learn by doing

## Code School's August Newsletter

Hello again, this is Gregg Pollack, the guy from Code School. I'm excited to fill you in on the latest news from our team.

**In this newsletter:**

- Rails for Zombies: Resurrection Announcement
- $40 coupon offer for Rails Best Practices
- Windy City Rails Tutorial
- Rails for Zombies, now in Español and coming in Italiano
- Get your company to join Code School, and save $
- "zombies".singularize => "zombie"

## Rails for Zombies: Resurrection Announcement

Great news for all you Zombie lovers, Rails for Zombies 2 is coming out next month! You couldn't get enough of it, so here comes more.

Mad Mimi
email marketing

madmimi.com

# TABLE OF CONTENTS


01 Creating and Sending Mail

02 Sending Attachments in Mail

03 The Asset Pipeline

04 Asset Tags

05 CoffeeScript

06 SCSS

07 JavaScript Manifest

# ASSET PIPELINE

## Rails 3.0

JUNK DRAWERS

# ASSET PIPELINE

## Rails 3.1

| | |
|---|---|
| app | Specific App Code |
| lib | My Shared Code |
| vendor | 3rd Party Code |

```
/assets/custom.js
/assets/rails.png
```

Looks in all paths

# ASSET TAG HELPERS

```
<%= javascript_include_tag "custom" %>
```

```
<script src="/assets/custom.js" type="text/javascript"></script>
```

```
<%= stylesheet_link_tag "style" %>
```

```
<link href="/assets/style.css" media="screen"
                                rel="stylesheet" type="text/css" />
```

```
<%= image_tag "rails.png" %>
```

```
<img alt="Rails" src="/assets/rails.png" />
```

In production

```
<img alt="Rails" src="/assets/rails-af27b6a414e6da000035031.png" />
```

FINGERPRINT

# ASSET PATHS IN STYLESHEETS

```
<img alt="Rails" src="/assets/rails-af27b6a414e6da000035031.png" />
```

Allows for better caching

app/assets/stylesheets/zombie.css

```
form.new_zombie input.submit {
  background-image: url(/assets/button.png);
}
```

NO CACHING

app/assets/stylesheets/zombie.css.erb

```
form.new_zombie input.submit {
  background-image: url(<%= asset_path('button.png') %>);
}
```

/assets/button-af27b6a414e6da000035031.png

# SCSS - SASSY CSS

**SCSS** is an extension to CSS3 adding nested rules, variables, mixins, selector inheritance, and more. **You get it by default in Rails 3.1.**

app/assets/stylesheets/zombie.css.scss.erb

```
form.new_zombie {
  border: 1px dashed gray;
}

form.new_zombie .field#bio {
  display: none;
}

form.new_zombie input.submit {
  background-image: url(<%= asset_path('button.png') %>);
}
```

ARRG! WE ARE REPEATING OURSELVES!!

**Lets do some nesting!**

# SCSS - SASSY CSS

## Using nested rules

app/assets/stylesheets/zombie.css.scss.erb

```
form.new_zombie {
  border: 1px dashed gray;

  .field#bio {
     display: none;
  }

  input.submit {
    background-image: url(<%= asset_path('button.png') %>);
  }
}
```

# COFFEESCRIPT

**CoffeeScript** is a programming langugage that compiles into JavaScript.
**You get it by default in Rails 3.1.**

app/views/zombies/_form.html.erb

```
<%= form_for(@zombie) do |f| %>
  ...
  <a href="#" id="bio-toggle">Show Bio Field</a>

  <div class="field" id="bio">
    <%= f.label :bio %><br />
    <%= f.text_area :bio %>
  </div>
  ...
<% end %>
```

**Objective**: When Show Bio Field is clicked, make bio field appear, and Show link disappear.

# COFFEESCRIPT

app/views/zombies/_form.html.erb

```
<a href="#" id="show-bio">Show Bio Field</a>

<div class="field" id="bio">
  <%= f.label :bio %><br />
  <%= f.text_area :bio %>
</div>
```

CSS

```
.field#bio {
    display: none;
}
```

First with regular jQuery JavaScript

app/assets/javascripts/zombies.js

```
$(document).ready(function(){
  $('#show-bio').click(function(event) {
    event.preventDefault();
    $(this).hide();
    $('.field#bio').show();
  }
}
```

# COFFEESCRIPT

app/assets/javascripts/zombies.js

```
$(document).ready(function(){
  $('#show-bio').click(function(event) {
    event.preventDefault();
    $(this).hide();
    $('.field#bio').show();
  }
}
```

Now with CoffeeScript

app/assets/javascripts/zombies.js.coffee

```
$(document).ready ->
  $('#show-bio').click (event) ->
    event.preventDefault()
    $(this).hide()
    $('.field#bio').show()
```

# REMEMBER THE SCAFFOLD?

```
$ rails g scaffold zombie name:string bio:text age:integer
```

Model

```
invoke    active_record
create    db/migrate/20110701230207_create_zombies.rb
create    app/models/zombie.rb
```

Routing

```
route     resources :zombies
```

Controller

```
create    app/controllers/zombies_controller.rb
```

View

```
create      app/views/zombies
create      app/views/zombies/index.html.erb
create      app/views/zombies/edit.html.erb
create      app/views/zombies/show.html.erb
create      app/views/zombies/new.html.erb
```

PLUS TESTS, HELPERS, AND ASSETS

# THE DEFAULT ASSET GENERATION

```
$ rails g scaffold zombie name:string bio:text age:integer
```

Model

```
invoke    active_record
create    db/migrate/20110701230207_create_zombies.rb
create    app/models/zombie.rb
```

Assets

```
invoke  assets
create    app/assets/javascripts/zombies.js.coffee
invoke    scss
create      app/assets/stylesheets/zombies.css.scss
```

From Gemfile

```
gem 'sass-rails'
```

```
gem 'coffee-script'
```

TO REMOVE, TAKE THEM OUT AND RERUN "BUNDLE INSTALL"

# THE DEFAULT ASSET GENERATION

```
app/assets/javascripts/zombies.js.coffee
```

```
app/assets/javascripts/tweets.js.coffee
```

```
app/assets/javascripts/brains.js.coffee
```

```
app/assets/javascripts/roles.js.coffee
```

Do we need to include each of these files?

```
<%= javascript_include_tag "zombies", "tweets", "brains", "roles" %>
```

**NO**, because of the application.js & Sprockets.

```
<%= javascript_include_tag "application" %>
```

In here is a Manifest of the JavaScript libraries we use.

# SPROCKETS

In here is a Manifest of the JavaScript libraries we use.

```
/app/assets/javascripts/application.js
```

```
//= require jquery
```
include the jQuery framework JavaScript

```
//= require jquery_ujs
```
include Rails specific unobtrusive JavaScript

```
//= require_tree .
```
include all files in this directory

Combines all files

```
<script src="/assets/application.js" type="text/javascript"></script>
```

In production

```
<script src="/assets/application-af27b6a414e6da000035031.js"
                                        type="text/javascript"></script>
```

To precompile all files into /public/assets/ (on your server)

```
$ rake assets:precompile
```

BY DEFAULT IT MINIFIES ALL CODE

# REQUIRING OTHER FILES

```
lib/assets/javascripts/shared.js.coffee
```

```
vendor/assets/javascripts/friend.js
```

```
/app/assets/javascripts/application.js
```

```
//= require jquery
//= require jquery_ujs
//= require shared
//= require friend
//= require_tree .
```

Can be used to specify order

# ASSET STYLESHEETS

/app/assets/stylesheets/application.css

```
/*
 *= require_self
 *= require_tree .
*/

form.new_zombie {
  border: 1px dashed gray;
}
```

Specifies where to insert content in this file

INCLUDED BEFORE EVERYTHING ELSE

# ASSET STYLESHEETS

/app/assets/stylesheets/application.css

```
/*
 *= require reset
 *= require_self
 *= require_tree .
*/

form.new_zombie {
  border: 1px dashed gray;
}
```

Included before the content in this file

```
<link href="/assets/application.css" media="screen"
                                rel="stylesheet" type="text/css" />
```

In production also going to be minified

```
<link href="/assets/application-af27b6414e6da0000.css" media="screen"
                                rel="stylesheet" type="text/css" />
```

http://guides.rubyonrails.org/



RailsGuides



# Asset Pipeline

This guide will cover the ideology of the asset pipeline introduced in Rails 3.1. By referring to this guide you will be able to:

- Understand what the asset pipeline is and what it does
- Properly organize your application assets
- Understand the benefits of the asset pipeline
- Adding a pre-processor to the pipeline
- Package assets with a gem

## Chapters

1. What Is The Asset Pipeline?
   - Main Features
   - What is fingerprinting and why should I care?
2. How to Use the Asset Pipeline
   - Asset Organization
   - Coding links to Assets
   - Manifest Files and Directives
   - Preprocessing
3. In Development
   - Debugging Assets
4. In Production
   - Precompiling assets
5. Customizing The Pipeline
   - CSS
   - Javascript

## 1 What Is The Asset Pipeline?

The asset pipeline provides a framework to concatenate and minify or compress Javascript and CSS assets. It also adds the ability to write these assets in other languages such as CoffeeScript, SCSS and ERB.

Prior to Rails 3.1 these features were added through third-party Ruby libraries such as Jammit and

# 5. RENDERING EXTREMITIES

# TABLE OF CONTENTS


01 Controller Rendering Options

02 Custom RESTful Routes

03 Rendering Custom JSON

04 Creating AJAX Links

05 AJAXified Forms

06 Sending Server JavaScript

07 AJAX Using JSON Data

# DEFAULT ACTION RENDERING

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def show
    @zombie = Zombie.find(params[:id])
  end
end
```

BY DEFAULT LOOKS FOR

app/views/zombies/show.html.erb

**Objective**: Respond to HTML and JSON

# DEFAULT RESPOND_TO BLOCK

**Objective**: Respond to HTML and JSON

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def show
    @zombie = Zombie.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @zombie }
    end
  end
end
```

**Objective**: If dead, render '/view/zombies/dead_again.html.erb'

# SPECIFYING A CUSTOM RENDER

**Objective**: If dead, render '/view/zombies/dead_again.html.erb'

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def show
    @zombie = Zombie.find(params[:id])
    respond_to do |format|
      format.html do
        if @zombie.decomp == 'Dead (again)'
          render :dead_again
        end
      end
      format.json { render json: @zombie }
    end
  end
end
```

IF NOT DEAD, RENDERS SHOW

**Objective**: Remove JSON, we don't need it.

# RESPONDS_TO JUST ONE THING

**Objective**: Remove JSON, we don't need it.

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def show
    @zombie = Zombie.find(params[:id])
    if @zombie.decomp == 'Dead (again)'
      render :dead_again
    end
  end
end
```

**Objective**: We ONLY need JSON

# IF WE ONLY NEED JSON

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def show
    @zombie = Zombie.find(params[:id])
    render json: @zombie
  end
end
```

## HTTP Status Codes

| | |
|---|---|
| 200 | :ok |
| 201 | :created |
| 422 | :unprocessable_entity |

DEFAULT

| | |
|---|---|
| 401 | :unauthorized |
| 102 | :processing |
| 404 | :not_found |

## Examples

```
render json: @zombie.errors, status: :unprocessable_entity
```

```
render json: @zombie, status: :created, location: @zombie
```

URL FOR NEW ZOMBIE

# CUSTOM ROUTE CHALLENGE

Create a new action in the Zombies controller which returns JUST decomp in JSON, and if "dead (again)" give status :unprocessable_entity

1. Add a new member route
2. Create a new action in the ZombiesController

# 1 Add a new member route

http://localhost:3000/zombies/4/decomp

config/routes.rb

```
match 'zombies/:id/decomp' => 'Zombies#decomp', :as => :decomp_zombie
```

```
resources :zombies do
  resources :tweets
  get :decomp, on: :member
end
```

SAME AS

Creates a custom RESTful Route

```
$ rake routes
```

```
decomp_zombie GET /zombies/:id/decomp {:action=>"decomp", :controller=>"zombies"}
```

Then we can use this route helper:

```
<%= link_to 'Get Decomposition Status', decomp_zombie_path(@zombie) %>
```

# 2 TYPES OF CUSTOM ROUTES

| | |
|---|---|
| :member | Acts on a single resource |
| :collection | Acts on a collection of resources |

| Route | URL | Route Helper |
|---|---|---|
| get :decomp, on: :member | /zombies/:id/decomp | decomp_zombie_path(@zombie) |
| put :decay, on: :member | /zombies/:id/decay | decay_zombie_path(@zombie) |
| get :fresh, on: :collection | /zombies/fresh | fresh_zombies_path |
| post :search, on: :collection | /zombies/search | search_zombies_path |

Examples:

```
<%= link_to 'Fresh zombies', fresh_zombies_path %>
```

```
<%= form_tag(search_zombies_path) do |f| %>
```

# CUSTOM ROUTE CHALLENGE

Create a new action in the Zombies controller which returns JUST decomp in JSON, and if "dead (again)" give status :unprocessable_entity

1. Add a new member route

```
resources :zombies do
  resources :tweets
  get :decomp, on: :member
end
```

2. Create a new action in the ZombiesController

# 2 Create a new action in the ZombiesController

/app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def decomp
    @zombie = Zombie.find(params[:id])
    if @zombie.decomp == 'Dead (again)'
      render json: @zombie, status: :unprocessable_entity
    else
      render json: @zombie, status: :ok
    end
  end
end
```


```
{
   "age":25,
   "bio":"I am zombified",
   "created_at":"2011-08-06T20:22:11Z",
   "decomp":"Fresh",
   "email":"zom@bied.com",
   "id":6,
   "name":"Eric",
   "rotting":false,
   "updated_at":"2011-08-06T20:22:11Z"
}
```


BUT WE WANT ONLY DECOMP!

# CUSTOMIZE JSON RESPONSE

```
@zombie.to_json(only: :name)
```

```
{ "name":"Eric" }
```

```
@zombie.to_json(only: [:name, :age])
```

```
{ "name":"Eric", age:25 }
```

```
@zombie.to_json(except: [:created_at, :updated_at, :id, :email, :bio])
```

```
{ "age":25, "decomp":"Fresh", "name":"Eric", "rotting":false }
```

```
@zombie.to_json(include: :brain, except: [:created_at, :updated_at, :id])
```

```
{
   "age":25,
   "bio":"I am zombified",
   "decomp":"Fresh",
   "email":"zom@bied.com",
   "name":"Eric",
   "rotting":false,
   "brain": {"flavor":"Butter","status":"Smashed","zombie_id":3}
}
```

HIDE THAT TOO

# SETTING THE JSON RESPONSE DEFAULT

```
@zombie.to_json
```

Set the default JSON Response

/app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  ...
  def as_json(options = nil)
    super(options ||
          {include: :brain, except: [:created_at, :updated_at, :id]})
  end
end
```

```
{
   "age":25,
   "bio":"I am zombified",
   "decomp":"Fresh",
   "email":"zom@bied.com",
   "name":"Eric",
   "rotting":false
   "brain": {"flavor":"Butter","status":"Smashed","zombie_id":3}
}
```

# CUSTOM ROUTE CHALLENGE

Create a new action in the Zombies controller which returns JUST decomp in JSON, and if "dead (again)" give status :unprocessable_entity

## 1 Add a new member route

```
resources :zombies do
  resources :tweets
  get :decomp, on: :member
end
```

## 2 Create a new action in the ZombiesController

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def decomp
    @zombie = Zombie.find(params[:id])
    if @zombie.decomp == 'Dead (again)'
      render json: @zombie.to_json(only: :decomp),
             status: :unprocessable_entity
    else
      render json: @zombie.to_json(only: :decomp)
    end
  end
end
```

```
{"decomp":"Fresh"}
```

# TABLE OF CONTENTS


01 Controller Rendering Options

02 Custom RESTful Routes

03 Rendering Custom JSON

04 Creating AJAX Links

05 AJAXified Forms

06 Sending Server JavaScript

07 AJAX using JSON data

# ADD AJAXIFIED DELETE LINK

When Delete is pressed do a fade out, instead of a page refresh.

1. Make the link a remote call
2. Allow the controller to accept the JavaScript call
3. Write the JavaScript to send back to the client

# 1 Make the link a remote call

/app/views/zombies/index.html.erb

```
<% @zombies.each do |zombie| %>
  <%= div_for zombie do %>
    <%= link_to "Zombie #{zombie.name}", zombie %>
    <div class="actions">
      <%= link_to 'edit', edit_zombie_path(zombie) %>
      <%= link_to 'delete', zombie, method: :delete %>
    </div>
  <% end %>
<% end %>
```

**Listing zombies**

| | |
|---|---|
| Zombie Joe | edit delete |
| Zombie Jim | edit delete |
| Zombie Bob | edit delete |
| Zombie Tony | edit delete |

# 1 Make the link a remote call

/app/views/zombies/index.html.erb

```
<% @zombies.each do |zombie| %>
  <%= div_for zombie do %>
    <%= link_to "Zombie #{zombie.name}", zombie %>
    <div class="actions">
      <%= link_to 'edit', edit_zombie_path(zombie) %>
      <%= link_to 'delete', zombie, method: :delete, remote: true %>
    </div>
  <% end %>
<% end %>
```

GENERATES

```
<a href="/zombies/5" data-method="delete"
                     data-remote="true" rel="nofollow">delete</a>
```

Rails will use unobtrusive JavaScript to do AJAX.

# ADD AJAXIFIED DELETE LINK

When Delete is pressed do a fade out, instead of a page refresh.

1. Make the link a remote call

```
<%= link_to 'delete', zombie, method: :delete, remote: true %>
```

2. Allow the controller to accept the JavaScript call
3. Write the JavaScript to send back to the client

# 2 Allow the controller to accept the JavaScript call

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def destroy
    @zombie = Zombie.find(params[:id])
    @zombie.destroy

    respond_to do |format|
      format.html { redirect_to zombies_url }
      format.json { head :ok }
      format.js
    end
  end
end
```

CAN RESPOND WITH JAVASCRIPT

# ADD AJAXIFIED DELETE LINK

When Delete is pressed do a fade out, instead of a page refresh.

1. Make the link a remote call

```
<%= link_to 'delete', zombie, method: :delete, remote: true %>
```

2. Allow the controller to accept the JavaScript call

```
respond_to do |format|
  format.js
end
```

3. Write the JavaScript to send back to the client

app/views/zombies/destroy.js.erb

```
$('#<%= dom_id(@zombie) %>').fadeOut();
```

# ADD AJAXIFIED DELETE LINK

When Delete is pressed do a fade out, instead of a page refresh.

# ADD AJAXIFIED CREATE ZOMBIE FORM

On the same Zombie index page, create an AJAX form which will create a new zombie and add it to the list.

1. Write format.js in the controller
2. Refactor the view and create the form
3. Create the JavaScript

# 1 Write format.js in the controller

app/controllers/zombies_controller.rb

```
class ZombiesController < ApplicationController
  def create
    @zombie = Zombie.new(params[:zombie])

    respond_to do |format|
      if @zombie.save
        format.html { ... }
        format.json { ... }
      else
        format.html { ... }
        format.json { ... }
      end
      format.js
    end
  end
end
```

# 2 Refactor the view and create the form

app/views/zombies/index.html.erb

```
<div id="zombies">
<% @zombies.each do |zombie| %>

  <%= div_for zombie do %>
    <%= link_to "Zombie #{zombie.name}", zombie %>
    <div class="actions">
      <%= link_to 'edit', edit_zombie_path(zombie) %>
      <%= link_to 'delete', zombie, method: :delete, remote: true %>
    </div>
  <% end %>

<% end %>
</div>
```

Lets refactor into a partial

# 2 Refactor the view and create the form

app/views/zombies/index.html.erb

```
<div id="zombies">
<% @zombies.each do |zombie| %>
  <%= render zombie %>
<% end %>
</div>
```

LOOKS FOR PARTIAL USING CLASS NAME

app/views/zombies/_zombie.html.erb

```
<%= div_for zombie do %>
  <%= link_to "Zombie #{zombie.name}", zombie %>
  <div class="actions">
    <%= link_to 'edit', edit_zombie_path(zombie) %>
    <%= link_to 'delete', zombie, method: :delete, remote: true %>
  </div>
<% end %>
```

# 2 Refactor the view and create the form

app/views/zombies/index.html.erb

```
<div id="zombies">
<% @zombies.each do |zombie| %>
  <%= render zombie %>
<% end %>
</div>
```

SAME THING

```
<div id="zombies">
  <%= render @zombies %>
</div>
```

# 2 Refactor the view and create the form

app/views/zombies/index.html.erb

```
<div id="zombies">
  <%= render @zombies %>
</div>

<%= form_for(Zombie.new, remote: true) do |f| %>
  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

# 3 Create the JavaScript

app/views/create.js.erb

```
$('div#zombies').append("<%= escape_javascript(render @zombie) %>");
$('div#<%= dom_id(@zombie) %>').effect('highlight');
```

app/views/zombies/index.html.erb

```
<div id="zombies">
  <%= render @zombies %>
</div>

<%= form_for(Zombie.new, remote: true) do |f| %>
  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

BUT WHAT ABOUT WHEN NAME IS BLANK?

# 3 Create the JavaScript

app/views/zombies/create.js.erb

```
<% if @zombie.new_record? %>
  $('input#zombie_name').effect('highlight', { color: 'red' });
<% else %>
  $('div#zombies').append("<%= escape_javascript(render @zombie) %>");
  $('div#<%= dom_id(@zombie) %>').effect('highlight');
<% end %>
```

MUST ADD JQUERY-UI

/app/assets/javascripts/application.js

```
//= require jquery

//= require jquery_ui

//= require jquery_ujs

//= require_tree .
```

# ADD AJAXIFIED CREATE ZOMBIE FORM

## Listing zombies

Zombie Jim edit delete

Zombie Tony edit delete

Name

Create Zombie

# ADD AJAXIFIED CUSTOM DECOMP FORM

Create a AJAX form that can set a custom decomposition phase on the zombie show page.

1. Create new custom route for our action
2. Define the form
3. Create action in the controller
4. Write the JavaScript to send back to the client

**Name:** Tony

**Decomposition Phase:** Rotten

**Custom Phase** Set

## 1 Create new custom route for our action

/config/routes.rb

```
resources :zombies do
  resources :tweets
  get :decomp, on: :member
  put :custom_decomp, on: :member
end
```

## 2 Define the form

/views/zombies/show.html.erb

VALUE WE WANT TO UPDATE

```
<strong>Decomposition Phase:</strong>
<span id="decomp"%><%= @zombie.decomp %></span>

<div id="custom_phase">
  <%= form_for @zombie, url: custom_decomp_zombie_path(@zombie),
                                                  remote: true do |f| %>
    <strong>Custom Phase</strong>
    <%= f.text_field :decomp %>
    <%= f.submit "Set" %>
  <% end %>
</div>
```

# ADD AJAXIFIED CUSTOM DECOMP FORM

Create a AJAX form that can set a custom decomposition phase on the zombie show page.

1. Create new custom route for our action

```
put :custom_decomp, on: :member
```

2. Define the form

```
<%= form_for @zombie, url: custom_decomp_zombie_path(@weapon),
                                                remote: true do |f| %>
```

3. Create action in the controller

4. Write the JavaScript to send back to the client

**Name:** Tony

**Decomposition Phase:** Rotten

**Custom Phase** Set

## 3 Create action in the controller

/app/controllers/zombies_controller.rb

```
def custom_decomp
  @zombie = Zombie.find(params[:id])
  @zombie.decomp = params[:zombie][:decomp]
  @zombie.save
end
```

DON'T NEED RESPOND_TO

## 4 Write the JavaScript to send back to the client

/app/views/zombies/advance_decomp.js.erb

```
$('#decomp').text('<%= @zombie.decomp %>').effect('highlight', {}, 3000);

<% if @zombie.decomp == "Dead (again)" %>
  $('div#custom_phase').fadeOut();
<% end %>
```

# ADD AJAXIFIED CUSTOM DECOMP FORM

Create a AJAX form that can set a custom decomposition phase on the zombie show page.

**Name:** Tony

**Decomposition Phase:** Fresh

**Custom Phase** Set

Edit | Back

# DO CUSTOM DECOMPOSITION, USING JSON WITH AN API

Do the same thing, but use JSON with client side JavaScript

1. Define the form
2. Modify the custom_decomp action
3. Write the client side JavaScript using CoffeeScript

**Name:** Tony

**Decomposition Phase:** Fresh

**Custom Phase** Set

**Custom Phase via JSON** Set

# 1 Define the form

/views/zombies/show.html.erb

```
<div id="custom_phase2">
  <%= form_for @zombie, url: custom_decomp_zombie_path(@zombie) do |f| %>
    <strong>Custom Phase via JSON</strong>
    <%= f.text_field :decomp %>
    <%= f.submit "Set" %>
  <% end %>
</div>
```

# 2 Modify the advance_decomp action

/app/controllers/zombies_controller.rb

```
def custom_decomp
  @zombie = Zombie.find(params[:id])
  @zombie.decomp = params[:zombie][:decomp]
  @zombie.save

  respond_to do |format|
    format.js
    format.json { render json: @zombie.to_json(only: :decomp) }
  end
end
```

# 3 Write the client side JavaScript using CoffeeScript

/app/assets/javascripts/zombies.js.coffee

```
$(document).ready ->
  $('div#custom_phase2 form').submit (event) ->
    event.preventDefault()

    url = $(this).attr('action')
    custom_decomp = $('div#custom_phase2 #zombie_decomp').val()

    $.ajax
      type: 'put'
      url: url
      data: { zombie: { decomp: custom_decomp } }
      dataType: 'json'
      success: (json) ->
        $('#decomp').text(json.decomp).effect('highlight')
        $('div#custom_phase2').fadeOut() if json.decomp == "Dead (again)"
```

# DO CUSTOM DECOMPOSITION, USING JSON WITH AN API

Do the same thing, but use JSON with client side JavaScript

**Name:** Tony

**Decomposition Phase:** Fresh

**Custom Phase** Set

**Custom Phase via JSON** Set

Edit | Back