

# Railties

why what & how?

Hi! I am

Jan

the internet knows me as

DefV

and I work for

Openminds

I'm here to talk about

Railties

<http://defv.be/railties.pdf>



Railties is a new Rails 3  
feature

# Rails 3

Railties is a new thing in Rails 3, and a quite important one in that.

what's SO HUGE about rails 3 isn't any of the shiny new features. it's the endless possibilities opened up by abstraction + modularization. 

10:01 AM Apr 20th via Tweetie

Retweeted by 2 people

 Reply  Retweet

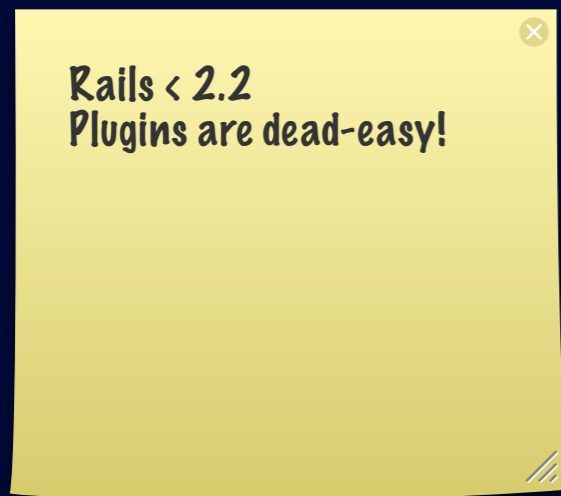


**svenfuchs**

Sven Fuchs

# Why Railties?

So why did we go to Railties? What was wrong with the current system? Lets have a small history lesson



# vendor/plugins

In the early days of Rails (<2, 2.1) THE way to share code was plugins. A plugin was easy to create and easy to install. The problem was you couldn't do good versioning and there was no dependency management. Voices went up to use Ruby's package management system Rubygems for Rails application extensions. So along came config.gem



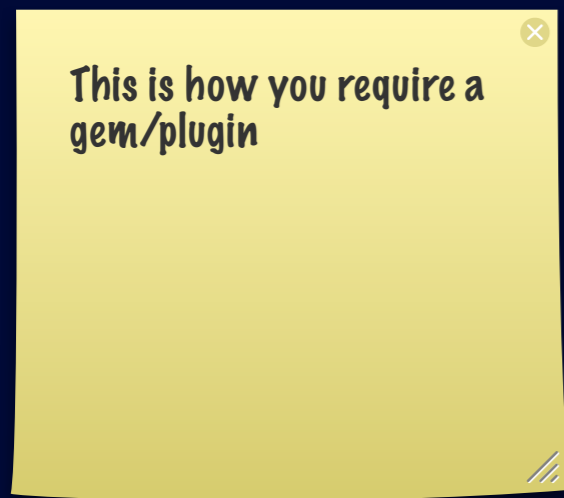
# config.gem

config.gem allowed you to setup your gem requirements in your general config file, along with their versions and whether to load in development or test or all. This was all great but still had some issues. The biggest of which was gem activation. If you had a gem that required “any” version of a certain gem, and later you load a gem (or Rails does) that requires a version lower than the one loaded, you’d get a nasty error. To solve this and other problems Yehuda Katz and Carl Lerche came up with Bundler



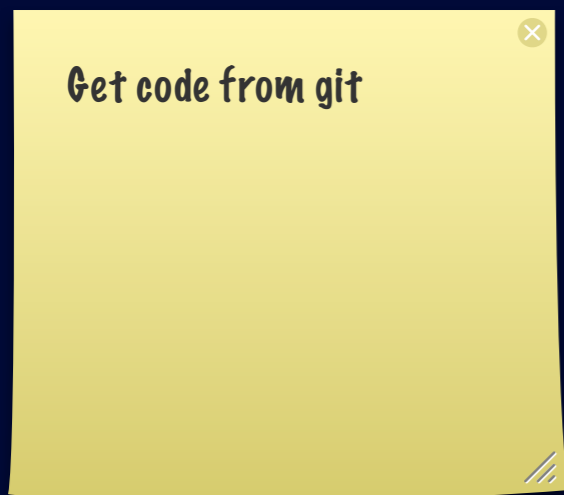
# Bundler

Bundler tries to solve all these problems by giving you a Gemfile where you define all dependencies of your application, with specific versions. Bundler then looks at all gems and figures out their dependencies, and installs them in a Bundle, with which you work.



gem 'will\_paginate'

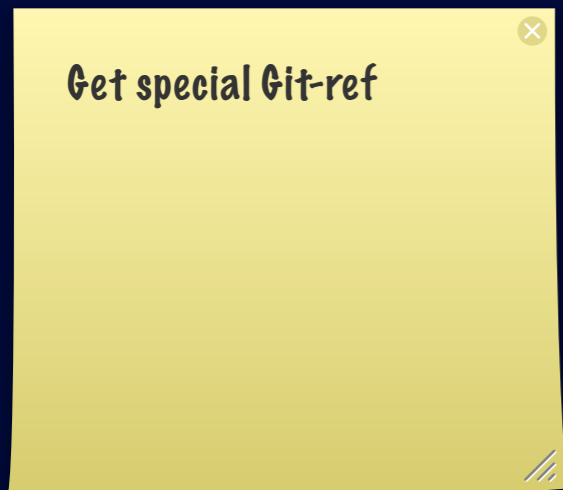
This is how you define a gem in your Gemfile



gem 'will\_paginate'

:git => [http://github.com/mislav/will\\_paginate.git](http://github.com/mislav/will_paginate.git)

You can even ask Bundler to get the Gem code from Git. If the given git repository has a gemspec present Bundler builds the gem and loads it



gem 'will\_paginate'

:git => 'http://github.com/mislav/will\_paginate.git'

:ref => 'rails3'

And since Rails 3 is still in Beta most developers just have a specific branch where they do their Rails3 fixes. You can also pass this in your Gemfile



# bundle install

This creates your gem-bundle, which you use from then on for all applications

# Rails specific code?

But that's just gem loading, what happens with code specifically meant for Rails?

```
vendor/plugins/  
will_paginate/  
init.rb
```

In plugins it was easy. A plugin was specifically used for Rails, and automatically the `init.rb` file is loaded.

`$GEMPATH/rails/  
init.rb`

Gems are a bit tougher, you can't use the `init.rb` because that might be used for some other purpose. Because of that you had to put your code in `rails/init.rb` when a gem got loaded.

# Rails::Railtie

Now we have a Railtie. We load the Railtie if Rails is loaded, and there you put your Rails specific code

```
rails raities_example.rb
1 class MyGem::Railtie < Rails::Railtie
2   config.my_gem = ActiveSupport::OrderedOptions.new
3
4   initialize 'mygem.extend.active_record' do
5     ActiveRecord::Base.extend(MyGem::AR) if defined?(ActiveRecord)
6   end
7 end
```

Line: 2 Column: 48 Ruby on Rails Soft Tabs: 2 MyGem::Railtie < Rails::Railtie

An example

ActionController::Railtie

ActiveRecord::Railtie

ActionView::Railtie

ActionMailer::Railtie

The great thing is that Rails has made all of its own components Railties. So if you want to use Rails with an other ORM you just don't load the ActiveRecord::Railtie, but your own, or you can swap ActionController::Railtie with your own controller-manager

# How?

So how do I create my own Railtie?

```
class MyGem::Railtie < Rails::Railtie
```

First we create our Railtie, and we inherit from Rails::Railtie. By doing that we let Rails know “We’re here, load us”

# config()

The config method allows us to set current config parameters to our liking, and also allows us to add a config part for our own Raitie

```
1 class MyGem::Railtie < Rails::Railtie
2   # Add a configurable option
3   config.my_gem = ActiveSupport::OrderedOptions.new
4
5   # Set :my_gem as templating engine
6   config.generator.template_engine = :my_gem
7
8   # Run on 1st request in production or
9   # on every request in development
10  config.to_prepare do
11    MyGem.setup
12  end
13 end
```

Line: 13 Column: 4 Ruby on Rails Soft Tabs: 2 MyGem::Railtie < Rails::Railtie

So here first I'm adding an config setting called `config.my_gem`. I can add settings now in my application, and my Railtie can read them and perform actions on them accordingly.

Next thing I'm doing is editing the `template_engine` generator to my own generator.

Last thin is setting up a `to_prepare`, which takes a block which will be executed at start on production and every time in development

# `initializer()`

The initializer method allows me to execute code when Rails gets initialized. I can choose to just run it on initialization, or before or after a certain action

```
config_example.rb
1 class MyGem::Railtie < Rails::Railtie
2   initializer "my_gem.load_middleware" do |app|
3     app.middleware.use MyGem::Rack
4   end
5
6   initializer "my_gem.extend.active_record" do
7     ActiveRecord.extend MyGem::AR if defined?(ActiveRecord)
8   end
9 end
```

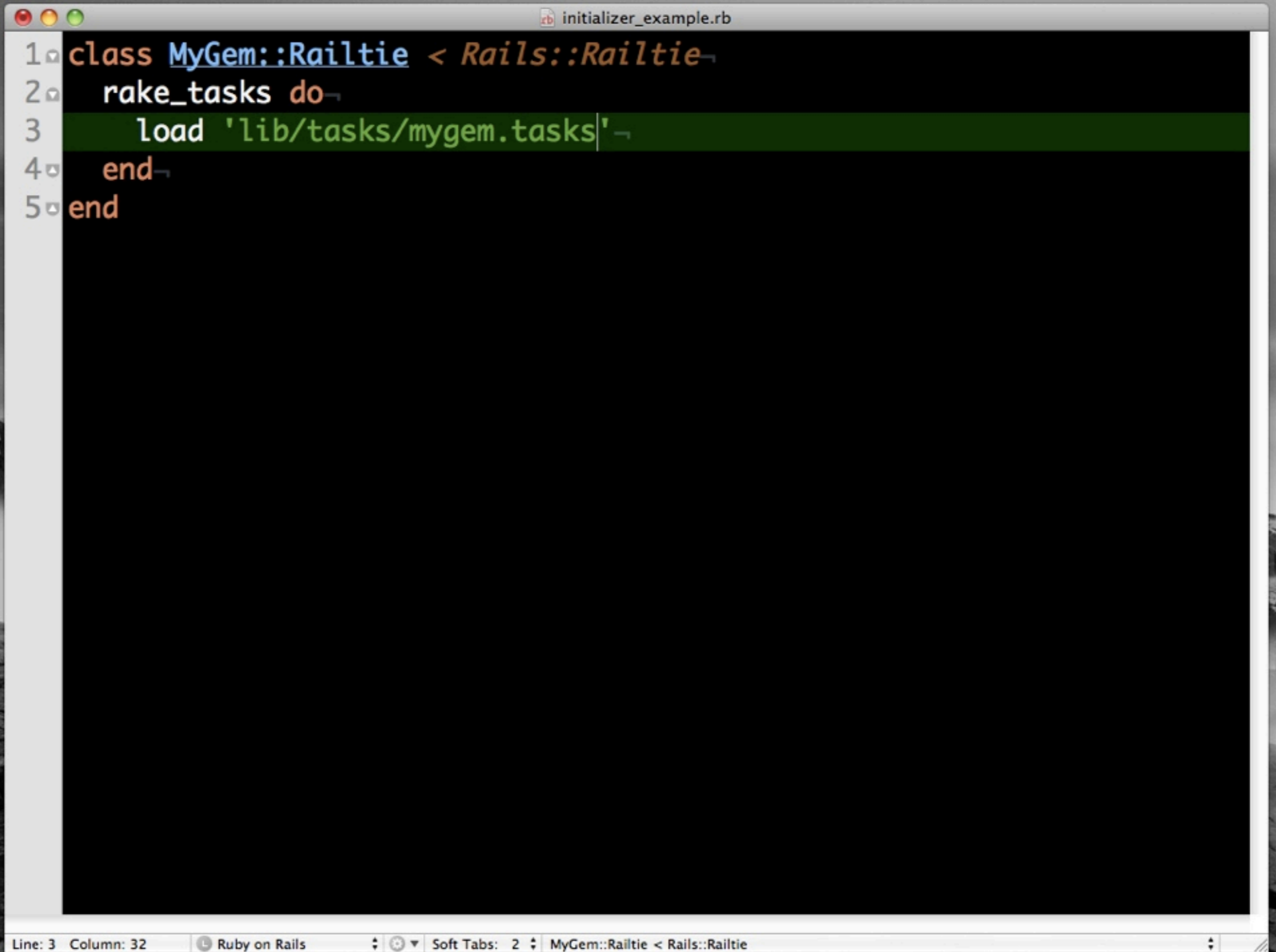
Line: 9 Column: 1 Ruby on Rails Soft Tabs: 2 MyGem::Railtie < Rails::Railtie

First thing you see is that I give my initializers a name. Convention is to use the name of your gem, a dot, and then the function it is performing.

So here I'm adding MyGem::Rack as middleware, I get yielded the app, which is the Rails app that gets started. In the second initializer I extend ActiveRecord with my own code if ActiveRecord is defined.

```
rake_tasks()
```

This function allows me to define which files should be loaded for Rake.



```
initializer_example.rb
1 class MyGem::Railtie < Rails::Railtie
2   rake_tasks do
3     load 'lib/tasks/mygem.tasks'
4   end
5 end
```

The screenshot shows a code editor window titled 'initializer\_example.rb'. The code defines a class `MyGem::Railtie` that inherits from `Rails::Railtie`. Inside the class, there is a `rake_tasks` block containing a `load` statement that loads the file `lib/tasks/mygem.tasks`. The code is as follows:

```
1 class MyGem::Railtie < Rails::Railtie
2   rake_tasks do
3     load 'lib/tasks/mygem.tasks'
4   end
5 end
```

The status bar at the bottom of the editor shows 'Line: 3 Column: 32', 'Ruby on Rails', 'Soft Tabs: 2', and 'MyGem::Railtie < Rails::Railtie'.

Here I load the task in `lib/tasks/mygem.tasks`.

# generators ( )

This function allows me to define which files should be loaded for Rake.

```
generators_example.rb
1 class MyGem::Railtie < Rails::Railtie
2   generators do
3     require "lib/my_gem/generator.rb"
4   end
5 end
```

Line: 3 Column: 21 Ruby on Rails Soft Tabs: 2 MyGem::Railtie < Rails::Railtie

```
# lib/my_gem.rb  
require 'railtie' if defined?(Rails)
```

There are two ways to make your Railtie available when your gem is loaded. This is the automatic way. We set it up so the Railtie gets loaded if the gem gets loaded and Rails is defined.

```
# Gemfile
gem 'my_gem', :require =>
  ['my_gem', 'my_gem/railtie']
```

The other way is to ask your users to require it themselves in their Gemfile.

# Rails::Engine

Next up are Rails Engines. They are basically Rails applications in your Rails application. They have routes, views, controllers, ..

`Rails::Engine < Rails::Railtie`



views, controllers,  
helpers

my\_engine/apps/\*

So in your engine you have an apps directory which can contain Views, Controllers, Helpers, Models, Metals, ... Anything you put in there gets added to the load path, so you can have Jobs or others in there too.

my\_engine/config/  
routes.rb

When your engine gets loaded, the routes in your config/routes.rb file automatically get added to the app's routes.

my\_engine/config/  
locales/ \*

Your locales are also added to the lookup path

```
my_engine/config/  
initializers/*
```

Initializers you define here get added to the regular initializers of your app

my\_engine/lib/  
tasks/ \*

Rake tasks also get loaded

```
1 class MyEngine < Rails::Engine
2   paths.app = "app"
3   paths.app.controllers = "app/controllers"
4   paths.app.helpers = "app/helpers"
5   paths.app.models = "app/models"
6   paths.app.metals = "app/metal"
7   paths.app.views = "app/views"
8   paths.lib = "lib"
9   paths.lib.tasks = "lib/tasks"
10  paths.config = "config"
11  paths.config.initializers = "config/initializers"
12  paths.config.locales = "config/locales"
13  paths.config.routes = "config/routes.rb"
14 end
```

# Rails::Plugin?

When you go through the documentation you'll see there's also a `Rails::Plugin` class. This isn't meant to be used for your gems, this is code that handles loading of legacy vendor/plugins code.

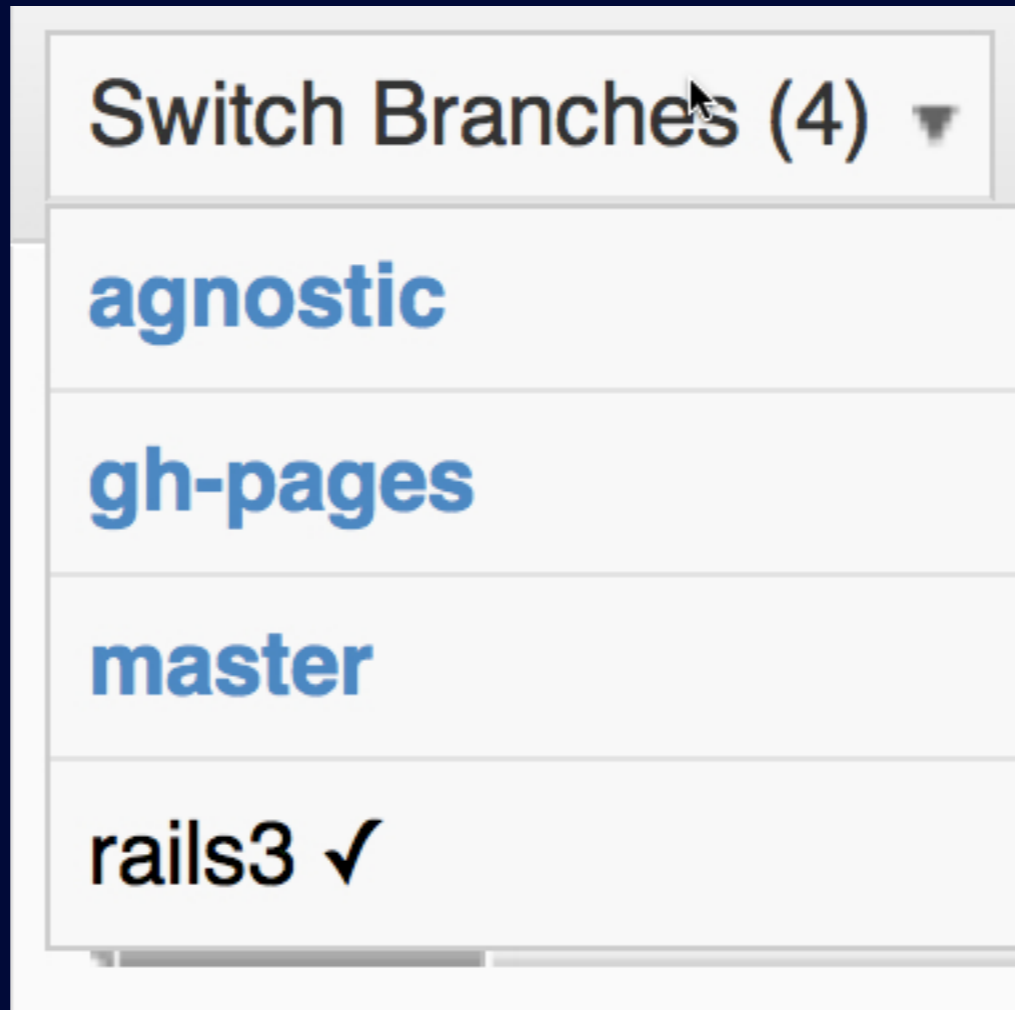
Compatibility?

```
rake_tasks_example.rb
1 # my_gem.rb
2 if defined?(Rails::Railtie)
3   require 'my_gem/railtie'
4 else
5   # Rails 2.3 way of loading file
6 end
```

Line: 6 Column: 4 Ruby on Rails Soft Tabs: 2

warning!

# Internal API changes



branches

# Is Your Plugin Ready For Rails 3?

**Rails 3** is just around the corner. It includes a whole **new API**. It's better, faster and stronger, but it might break existing plugins and gems. Use this site to quickly **find and verify** if the plugin or gem you want to use **works with Rails 3**. We track **340 plugins** with **538 versions** through **2,436 opinions**, add yours!

**Browse** the plugins, or **sign up** to get involved.

## Recent Activity

-  **BushyMark** is unsure if **globalize2 0.2.1** works with "Ruby 1.9". [Help out.](#)

---

-  **BushyMark** is unsure if **globalize2 0.2.1** works with "JRuby". [Help out.](#)

---

-  **BushyMark** is unsure if **globalize2 0.2.1** works with "Thread Safety". [Help out.](#)

---

-  **BushyMark** is unsure if **globalize2 0.2.1** works with "Rails 3". [Help out.](#)

---

-  **BushyMark** added version **0.2.1** for **globalize2**.

## Latest Versions

<b>globalize2</b> (0.2.1)	<b>Rails 3 ?</b>
<b>meta_where</b> (0.3.1)	<b>Rails 3 ✓</b>
<b>pivotal-tracker</b> (0.1.2)	<b>Rails 3 ?</b>
<b>restfulie</b> (0.7.1)	<b>Rails 3 ?</b>
<b>Bushido</b> (master)	<b>Rails 3 ✗</b>
<b>master_may_i</b> (0.6.2)	<b>Rails 3 ✗</b>

Questions?