Refinery CMS

Getting Started

This guide covers getting up and running with Refinery CMS. After reading it, you should be familiar with:

- Installing and creating a new Refinery site
- Changing the front end design of a Refinery site
- Extending Refinery's functionality with custom extensions

Chapters

Edit this guide on Github

This guide is based on Refinery CMS 2.1.0 so some of the code shown here may not work in earlier versions of Refinery.

1 Guide Assumptions

This guide is designed for beginners who want to get started with a Refinery CMS site from scratch. It does not assume that you have any prior experience with Refinery. However, to get the most out of it, you need to have some prerequisites

installed:

- The **Ruby** language version 1.9.3 or higher
- A working installation of the **SQLite3 Database**
- A working installation of **ImageMagick**

If you don't already have these things, then you will need to follow the **Installing Refinery Prerequisites** guide.

It is recommended that developers use RVM to manage Ruby versions and gems. Once RVM is installed on your system, and you've created a gemset for the project, create an .rvmrc file in the site's root directory. Its contents should look something like:

```
rvm use --create ruby-1.9.3@refinery
```

If you are not using an RVM gemset and you have Rails 4 installed, be sure to read the **Rails Application Templates** section below.

Refinery is a Ruby on Rails web application. If you have no prior experience with Rails, you will find a very steep learning curve diving straight into Refinery. There are some good free resources on the Internet for learning Rails, including:

- **Getting Started with Rails**
- **Rails for Zombies**

2 What is Refinery CMS?

Refinery CMS, often shortened to Refinery, is an open source content management system written in Ruby as a Ruby on Rails web application with JQuery used as the JavaScript library. Refinery runs on Rails 3.2.

Refinery differs from similar projects by targeting a non-technical end user and allowing the developer to create a flexible website rapidly by staying as close as possible to the conventions of the Ruby on Rails framework.

The Refinery philosophy includes several guiding principles:

- **"The Rails Way" where possible** – Refinery embraces conventions used in Rails, allowing any Rails programmer to leverage existing knowledge to get up and running quickly.
- **End user focused interface** – Refinery's user interface is simple, bright and attractive so end users feel invited and not overwhelmed.
- **Awesome developer tools** – Refinery makes it easy for developers to add functionality and change the front end look and feel.
- **Encourage and Help Others** – Refinery has an active community on Google Groups and IRC. If you ever have a question there is someone able and willing to assist.

2.1 Refinery's architecture

Refinery is comprised of several Rails Engines. Each extension acts like a mini Rails application with its own routes and views. Refinery is architected like this so that it keeps out of the way of any custom development you will do in the `/app` directory.

2.2 Core extensions

The extensions Refinery comes with are:

- **Authentication** – manages users and sessions within Refinery.
- **Core** – contains default layouts, views, javascripts and CSS. This extension also has an extension API for extending Refinery and everything Refinery needs to hook into Rails.
- **Dashboard** – shows you what's recently been updated.
- **Images** – handles image upload, insertion and processing images using **Dragonfly**.
- **Pages** – allows you to manage pages including the structure of your site displayed in the front end.
- **Resources** – handles file upload and storage.

3 Creating a new Refinery project

If you follow this guide, you'll create a Refinery site called `rickrockstar` that will have a custom design and an events extension to allow Rick to tell his fans when his next gig is.

Before you can start building this site, you need to make sure that you have Refinery itself installed.

3.1 Installing Refinery

There are two popular ways to install Refinery: RubyGems and Rails Application Templates. We will discuss both and how you can get up and running with either.

3.1.1 RubyGems

The easiest way to install Refinery is to take advantage of RubyGems.

```
$ gem install refinerycms
```

RootFile

This step may take some time to load as it needs to download and install all the ruby gems Refinery depends on.

If `gem install refinerycms` fails with an error message saying `ERROR: While executing gem ... (Gem::ImpossibleDependenciesError)`, run `gem install rails -v=3.2.15` then `gem install refinerycms`

RootFile

If you're working on Windows, you should be aware that the vast majority of Refinery development is done in Unix environments. If at all possible, we suggest that you develop on a Linux based operating system.

3.1.2 Rails Application Templates

These application templates are another very easy way to install Refinery, and allow for a great deal of control of your installation. You can create a new Refinery application by typing:

```
$ rails new rickrockstar -m http://refinerycms.com/t/2.1.0
```

RootFile

As **Refinery CMS is not compatible with Rails 4** you may also need to specify the Rails version in case you are not using an RVM gemset:

```
$ rails _3.2.15_ new rickrockstar -m http://refinerycms.com/t/2.1.0
```

RootFile

If you create an application from the template, be sure to skip running `refinerycms rickrockstar` as listed in the next section—the template will have completed this step.

3.2 Creating a Refinery application

The best way to use this guide is to follow each step as it happens. No code or step needed to make this example application has been left out, so you can literally follow along step-by-step.

To begin, open a terminal, navigate to a folder where you have rights to create files, and type:

```
$ refinerycms rickrockstar
```

RootFile

This will create a new Rails application with Refinery built in called RickRockStar in a directory called `rickrockstar`. It also automatically runs `bundle install` which will find and install all Refinery's ruby gem dependencies. Finally, it creates your database and seeds it with some basic defaults to get you started.

In this guide we are using an SQLite3 database for data storage, because it is a zero-configuration database that works without effort. Refinery also supports MySQL and PostgreSQL "out of the box". For important applications, you may wish to use one of these other database systems.

Refinery will create a folder in your working directory called `rickrockstar`. Switch to this folder:

```
$ cd rickrockstar
```

DeclareReference

Open up that folder and explore its contents. You'll notice what you have is a very standard Rails application.

4 Hello, Refinery!

One of the traditional places to start with a new project is by getting some text up on screen quickly. To do this, you need to get your Refinery application server running.

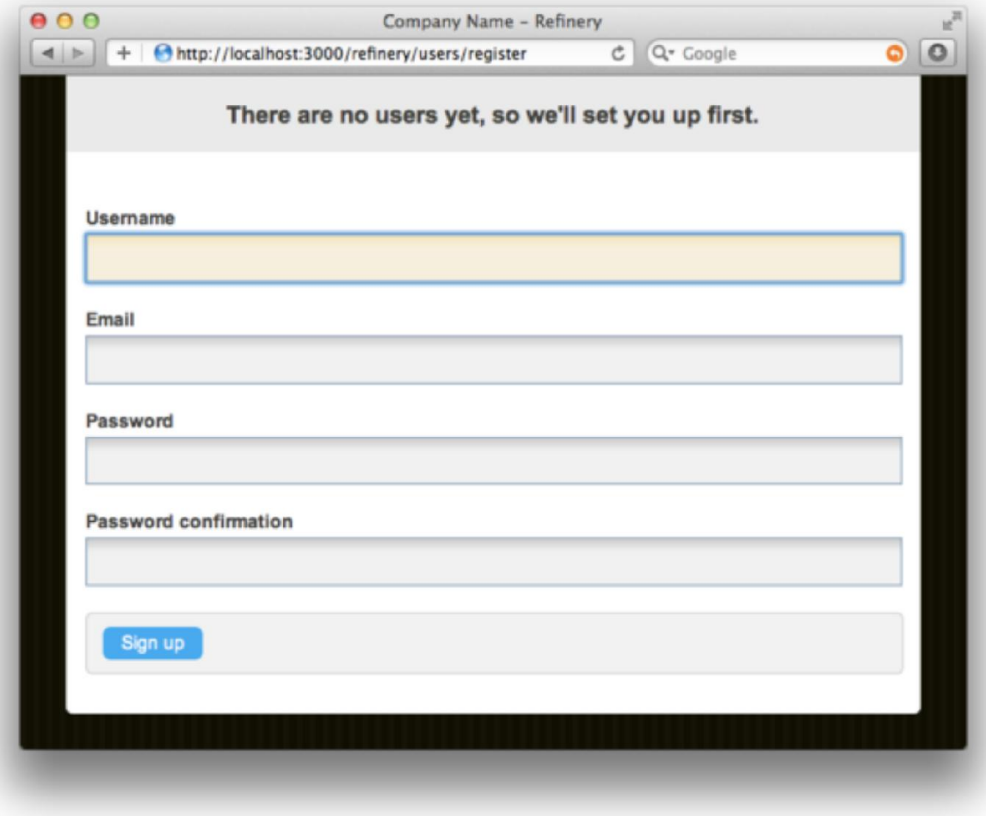
4.1 Starting up the Web Server

You actually have a functional Refinery application already installed. To see it, you need to start a web server on your development machine. You can do this by running:

```
$ rails server
```

RootFile with cd reference referenced

This will fire up an instance of the built-in Rails web server by default (called WEBrick). To see your application in action, open a browser window and navigate to **http://localhost:3000/refinery**. You should be greeted with a screen prompting you to create your first Refinery user.



To stop the web server, hit `Ctrl+C` in the terminal window where it's running.

If you do not see this page, but rather see a default page for Rails, ensure you have removed

`public/index.html`. The installer should have removed this file for you, but in certain circumstances, it may persist.

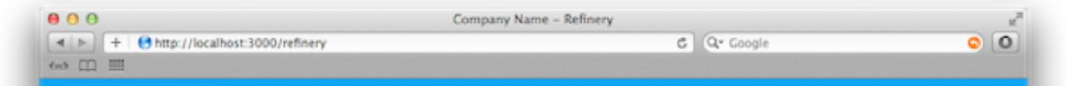
If you encounter a routing error afterwards, run rails generate refinery:cms —fresh-installation

If you see this screen it means you have set up everything correctly and your Refinery CMS site is up and running.

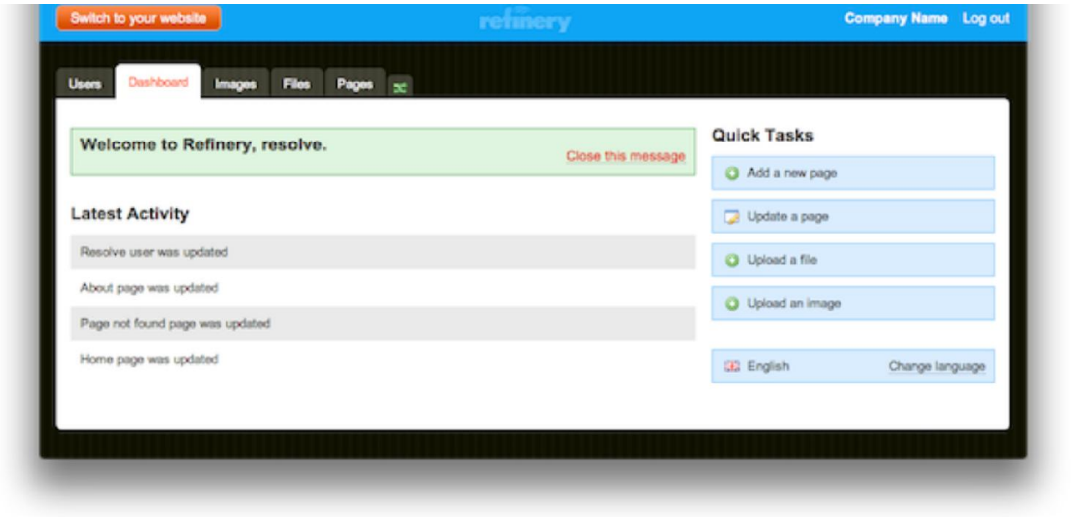
4.2 Setting up your first user

Fill out the sign up form to create your first user. This will be the super user of Refinery, meaning this user will be able to create other users later on.

Once you've created your first user you'll see Refinery's "Dashboard".



4.3 Setting Your Site Name



You'll need to set your Site Name; it's used in several spots on the CMS to give you nice branding (for instance, in the blue header at the top of the page and in the footer of your site).

To do this, you'll have to edit `config/initializers/refinery/core.rb`. Look for the line that begins:

```
# config.site_name = "Company Name"
```

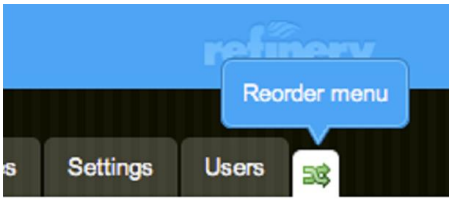
ChangeFileRegex

The `#` character prefixing the line is a comment character. The configuration options in this file (and in other Refinery initializer files) are all commented out, because these options are already set deep inside of Refinery. If you uncomment a line, Refinery will prefer the value you set inside these initializers. Go ahead and remove the `#` character plus the space before the word 'config', and then change "Company Name" to "Rick Rock Star". Make sure you save, and then restart your server (if you're using the built-in Rails server, hit `Ctrl+C`, and then type `rails server` again.

Many parts of Refinery can be customized by changing the options contained within the `config/initializers/refinery/` folder. As you add extensions to Refinery, more files will be created here specific to the extensions you install.

4.4 Explore Refinery

Now you're setup, click around the various tabs in the backend and become familiar with what comes out of the box.



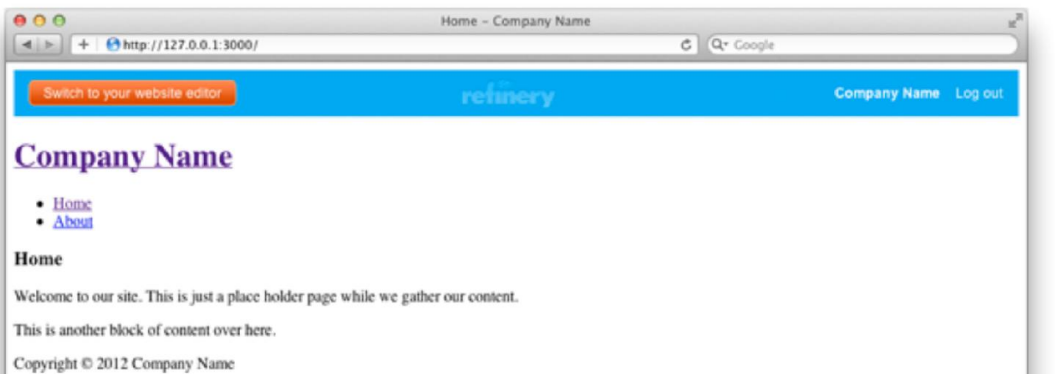
While exploring, one of the first things to do is reorder the tabs to reflect what you will be working on the most. Click the tab with the two green arrows, and then drag the backend tabs – for example, "Users" or "Images" – into an order which makes the most sense to you.

Click the reorder button again when you're done to save.

4.5 Switching to your front end

You're currently in the back-end Refinery site editor. To see the front-end site, click "Switch to your website".

As you can see, Refinery is already displaying a basic menu and layout ready for you to customise.



5 Customising the Design

The layout Refinery provides out of the box is very bare. We'll now guide you through

customising the front end design to give Rick the beautiful site we promised.

5.1 Overriding your first view

By default Refinery has a range of views built in to display the front end site you currently have. But more times than often you want to customise them with your own layout and design.

Overriding Refinery views is a common pattern which is worth keeping in mind at all times. If Refinery isn't displaying something how you'd like, just override it.

If you request `http://localhost:3000/about`, this maps by default to Refinery's `pages_controller.rb` show action.

So as you would expect according to Rails convention, the view for this action is located in `app/views/refinery/pages/show.html.erb`. You won't be able to see this file because Refinery is providing it for you. Next, let's override that view and replace it with our own.

Overriding a file simply copies the file from Refinery's code into your `app/` folder. Many people are confused as to what can be overridden initially. Any controller, model, view, javascript, or stylesheet from any installed extension can be overridden, but the most commonly overridden ones are those in the **refinery** folder.

5.2 Overriding your first view

Refinery comes with a rake task called `refinery:override` which allows you to copy files out of Refinery and into your own application to change. To see a list of possible commands simply run `rake refinery:override` in the console. Let's override the pages show view:

```
$ rake refinery:override view=refinery/pages/show
    create  app/views/refinery/pages/show.html.erb
```

RootFile with cd referenced

Now edit `app/views/refinery/pages/show.html.erb` and it will look like this:

```
<%= render '/refinery/content_page' %>
```

ChangeFileLines

Refinery has a `content_page` partial it uses just to get you started. But we'll soon remove this and use our own ERB view instead.

5.3 Rendering content using `@page`

Every view in Refinery has an instance variable called `@page` available. The best way to explain how this works is just to show you.

Replace the contents of `app/views/refinery/pages/show.html.erb` with this:

```
<section id='body'>
  <%=raw @page.content_for(:body) %>
</section>
<section id='side_body'>
  <%=raw @page.content_for(:side_body) %>
</section>
```

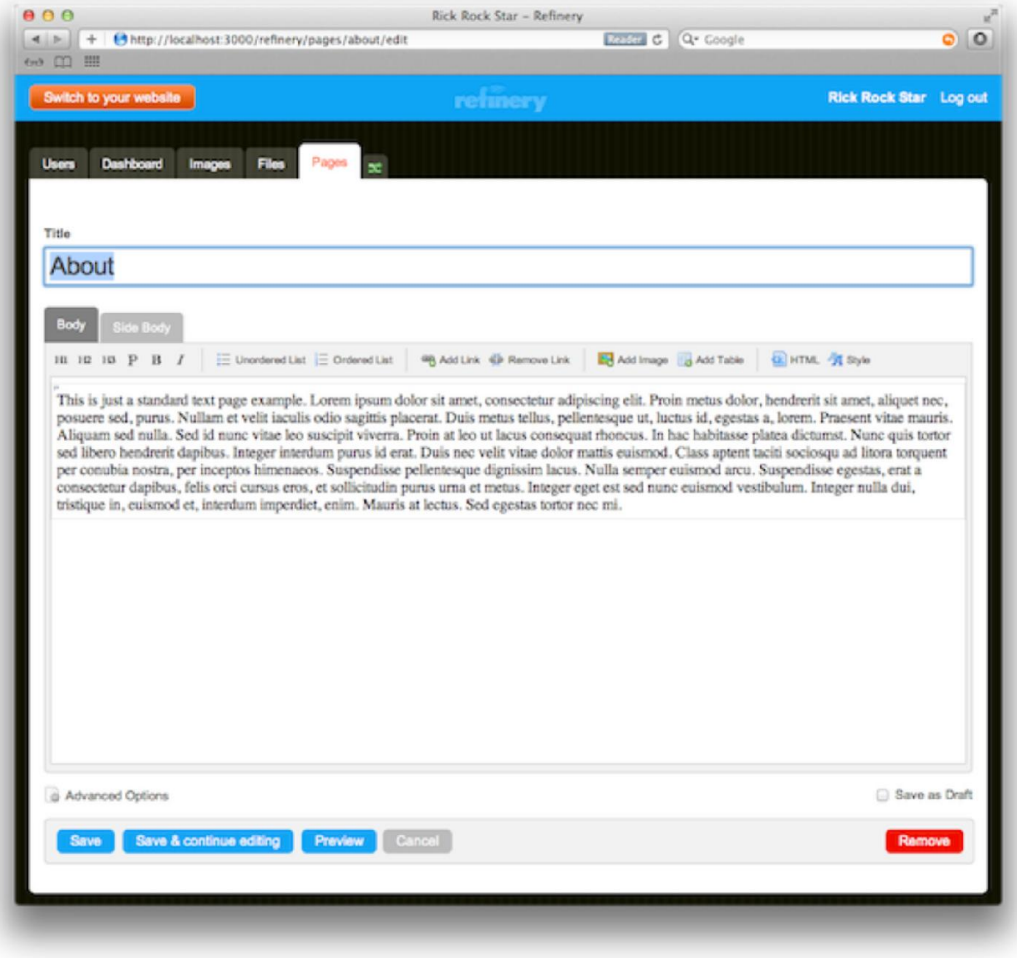
ChangeFileLines

As you can see we're going to render a view with some HTML5 tags and along with some content coming from the CMS (those are the lines that mention `@page`).

`@page` has what we call `PageParts` associated with it. To see what we mean, go to **http://localhost:3000/refinery/pages**

and then click “Edit” on the page titled “About”.

When you edit the About page you’ll see something like this:



You’ll notice two tabs on the page: “Body” and “Side Body”. These are `PageParts`, or in other words, a single piece of content attached to this page that you can render in your view. There is a “Body” tab with some content on this screen. To render that same content in your view, put:

```
<%=raw @page.cor
```

5.4 Styling your views

As mentioned earlier, a key principle in Refinery is to stick to “The Rails Way” where possible. This is apparent in the way you style your views too. You style your site exactly how you would style any Rails 3.1 application, using `app/assets/stylesheets/application.css.scss`.

Open up `app/assets/stylesheets/application.css.scss` and add this:

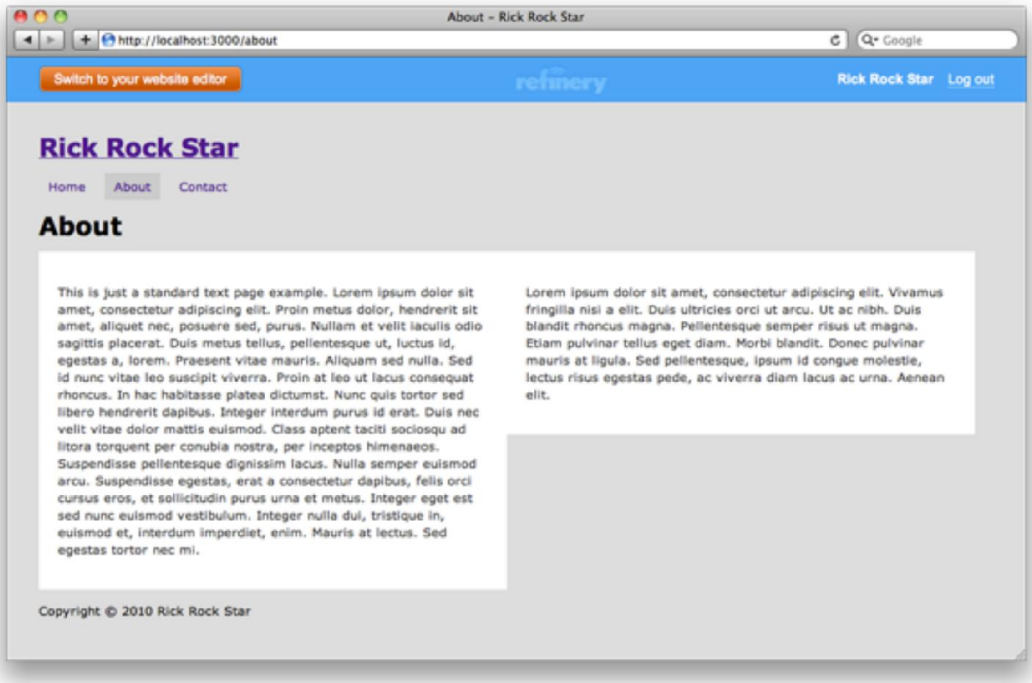
```
body {
  background: #DDD;
  font-family: Verdana;
  font-size: 13px;
  line-height: 18px;
}

#body, #side_body {
  float: left;
  width: 45%;
  background: white;
  color: #333;
  padding: 20px;
}

#menu ul {
  list-style: none;
  padding: 0px;
```

InsertIntoFile

```
  li {
    float: left;
    margin-right: 10px;
    a {
      display: block;
      padding: 5px 10px;
      text-decoration: none;
    }
    &.selected a, a:hover {
      background: #CCC;
    }
  }
}
```

You can add CSS for specific pages by using their slug with -page. The default for the home page is ‘home-page’.

Now when you view your front end at **http://localhost:3000** you’ll notice your site has a grey background, with a horizontal menu and two white content areas.



5.5 What we just did

We just overwrote the `pages/show` view and replaced it with our own version. We learnt how to use `@page` to display content entered in the backend.

Finally we added a (super) simple style that changes the colour of the background and uses our `pages/show` view to split into two even columns.

6 Extending Refinery with your first Engine

6.1 The Anatomy of an Engine

Think of a Refinery extension as a miniature Rails application running in your `vendor/extensions` directory. Each extension specifies its own routes in its config directory and has its own views and controllers in its own app directory. Engines can even serve up their own images, stylesheets and javascripts by utilizing the asset pipeline, which got introduced in Rails 3.1.

6.2 Generating an extension

Refinery ships with an extension generator that makes adding your own functionality a breeze. It works just like the Rails scaffold generator.

RootFile with cd referenced

```
$ rails generate refinery:engine singular_model_name attribute:type [attribute:type ...]
```

to see all the options supported by the `refinery:engine` generator just run `rails g refinery:engine`.

RootFile with cd referenced

Here is a list of the different field types and what they give you:

field type	description
text	a multiline visual editor
resource	a link which pops open a dialog which allows the user to select an existing file or upload a new one
image	a link which pops open a dialog which allows the user to select an existing image or upload a new one
string and integer	a standard single line text input

If you remember, we told Rick that we'll give him an area to post up events he'll be at. Although he could technically create a new page in Refinery to add this content there, areas that have special functionality are much better suited as an extension.

Rick is going to want to enter the following information about each event:

- The title
- The date of the event
- A photo
- A little blurb about the event.

RootFile with cd referenced

Run this command to generate the events extension for Rick:

```
$ rails generate refinery:engine event title:string date:datetime photo:image blurb:text
```

This results in the following:

```
create  vendor/extensions/events/app/controllers/refinery/admin/events_controller.rb
      create  vendor/extensions/events/app/controllers/refinery/events_controller.rb
      create  vendor/extensions/events/app/models/refinery/event.rb
      create  vendor/extensions/events/app/views/refinery/admin/events/_actions.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/_form.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/_events.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/_records.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/_event.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/_sortable_list.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/edit.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/index.html.erb
      create  vendor/extensions/events/app/views/refinery/admin/events/new.html.erb
      create  vendor/extensions/events/app/views/refinery/events/index.html.erb
      create  vendor/extensions/events/app/views/refinery/events/show.html.erb
      create  vendor/extensions/events/config/locales/en.yml
      create  vendor/extensions/events/config/locales/es.yml
      create  vendor/extensions/events/config/locales/fr.yml
      create  vendor/extensions/events/config/locales/lolcat.yml
      create  vendor/extensions/events/config/locales/nb.yml
      create  vendor/extensions/events/config/locales/nl.yml
      create  vendor/extensions/events/config/routes.rb
      create  vendor/extensions/events/db/migrate/20111031210430_create_events.rb
      create  vendor/extensions/events/db/seeds.rb
      create  vendor/extensions/events/lib/generators/refinery/events_generator.rb
      create  vendor/extensions/events/lib/refinerycms-events.rb
      create  vendor/extensions/events/lib/tasks/events.rake
      create  vendor/extensions/events/readme.md
      create  vendor/extensions/events/refinerycms-events.gemspec
      ...
------------------------
```

```
Now run:
bundle install
rails generate refinery:events
rake db:migrate
rake db:seed
Please restart your rails server.
-------------------------
```

As the output shows next run:

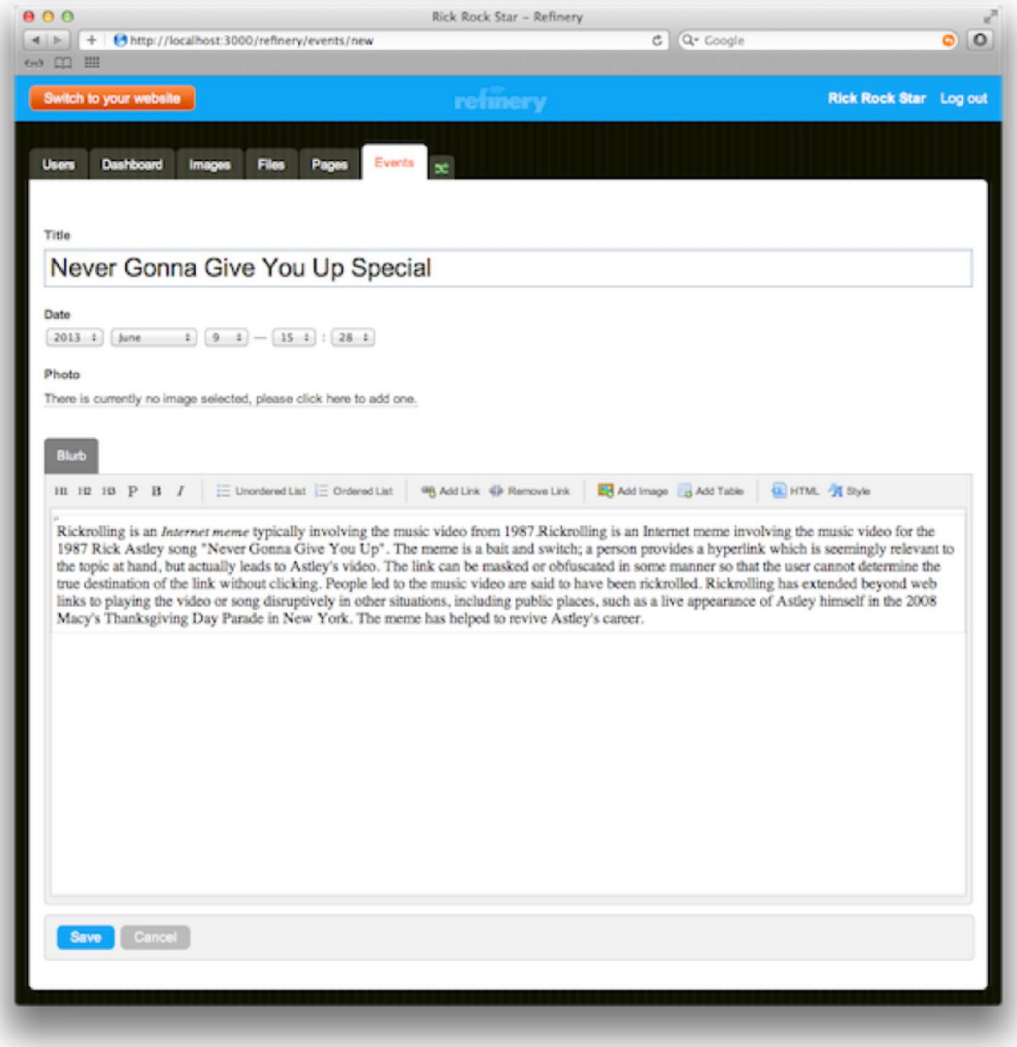
```
$ bundle install
$ rails generate refinery:events
$ rake db:migrate
$ rake db:seed
```

RootFile with cd referenced

A `refinery:events` generator is created for you to install the migration to create the events table.

When new extensions are added it's a good idea to restart your server for new changes to be loaded in.

Now go to the backend of your Refinery site (**http://localhost:3000/refinery**) and you'll notice a new tab called "Events". Click on "Add new event" and you'll see something like this:



You'll see the entire form has been generated for you based off the field types you specified when generating the events section. The blurb has a visual editor, the date field is a date picker and the photo allows you to pick or upload a new photo from a built-in Refinery dialog.

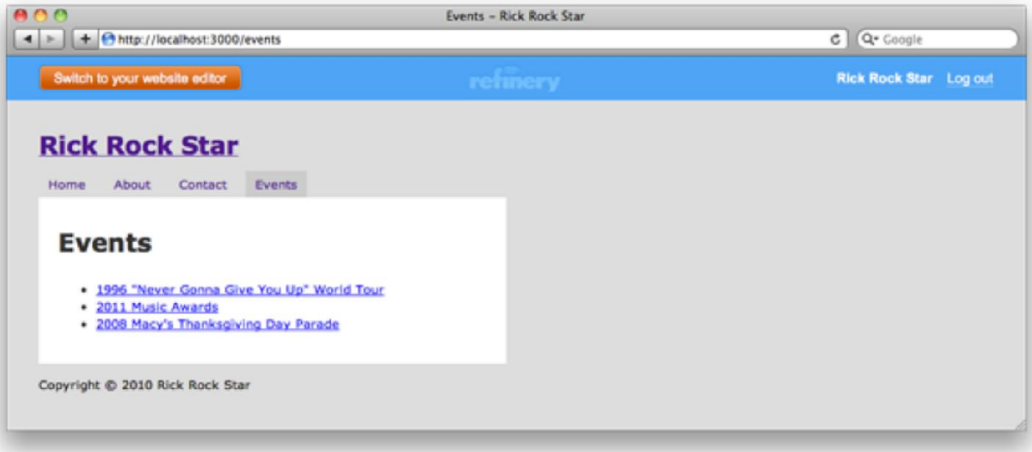
Add a couple of mock events to your events extension.

Now click on "Switch to your website", and navigate to

http://localhost:3000/events

You'll notice not only has Refinery generated the backend "Events" tab but also a new menu item called "Events" and two new front-end views, `index.html.erb` and `show.html.erb`, located in `vendor/extensions/events/app/views`

`/refinery/events/` for you to customise.



As you can see, Refinery makes it insanely easy to quickly add new extensions to manage various areas of a site.

But I've noticed one problem. The "2011 Music Awards" is showing up in the middle when it makes more sense to order the events with the latest event at the top. To fix this we need to understand what's happening under the hood of a Refinery extension. Let's dive in.

6.3 Testing your extension

There is a separate guide which covers this subject found at **How to Test Your Engine**.

6.4 Crudify: The Backbone of Refinery Engines

Any Refinery extension, even the built-in ones, that focus on Create, Read, Update and Delete are driven by crudify. Crudify is a highly reusable module included with Refinery that gives you all the standard CRUD actions as well as reordering, searching and paging.

Open up `vendor/extensions/events/app/controllers/refinery/events/admin/events_controller.rb` and look at its contents:

```
module Refinery
  module Events
    module Admin
      class EventsController < ::Refinery::AdminController

        crudify :'refinery/events/event', :xhr_paging => true

      end
    end
  end
end
```

ChangeFileLines

Most of the time, crudify's defaults are bang on, but if you need to, you can easily customise how it works.

By default `crudify` assumes your records will be sortable. But events should not be manually sortable; it makes more sense to order them by their event date. Update the contents of the file to this:

```
module Refinery
  module Events
    module Admin
      class EventsController < ::Refinery::AdminController

        crudify :'refinery/events/event', :xhr_paging => true,
                                          :order => "date DESC",
```

ChangeFileLines

```
                                        :sortable => false

      end
    end
  end
end
```

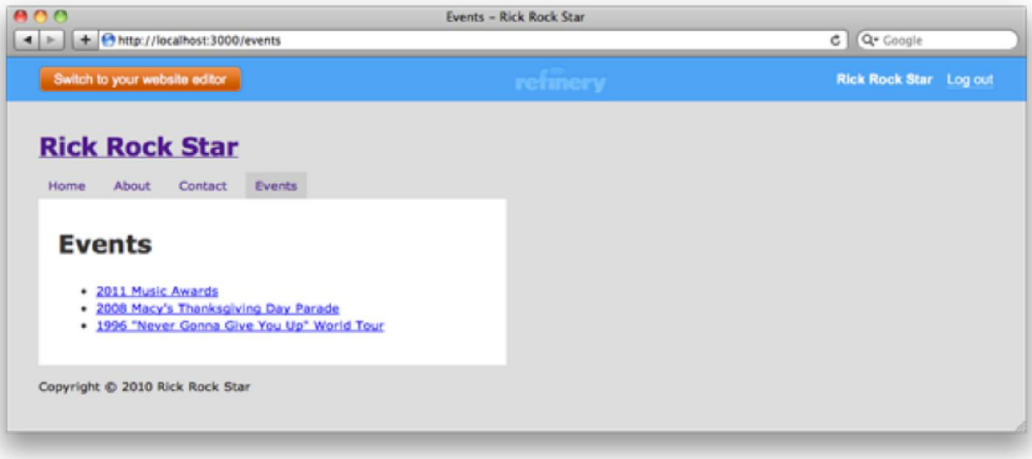
This will tell `crudify` to sort by our event date field and to turn off manual sorting by the user.

Finally edit `vendor/extensions/events/app/controllers/refinery/events/events_controller.rb` and replace the `find_all_events` method with this one:

```
module Refinery
  module Events
    class EventsController < ::ApplicationController

      # code

      protected

        def find_all_events
          # Order by event date
          @events = Event.order("date DESC")
        end

        # code

    end
  end
end
```

ChangeFileRegex

Now when you look at **http://localhost:3000/events** you'll notice they're now being sorted by the event date.



7 What's Next?

Now that you've made your first Refinery application with a custom events extension, you should feel free to update it and experiment on your own. But you don't have to do everything without help.

If you need assistance getting up and running with Refinery, follow the **How to get help with Refinery Guide**.

Follow us on Twitter

- **Showcase**
- **Hosting**
- **Guides**
- **Feedback**
- **Extensions**

- **Download**
- **Community**
- **Blog**
- **About**
- **Home**