Documentation for
RSpec Rails 2.8

RSpec Rails 2.8

rspec-rails extends Rails' built-in testing framework to support rspec examples for requests, controllers, models, views, helpers, mailers and routing.

Rails-3

rspec-rails-2 supports rails-3.0.0 and later. For earlier versions of Rails, you need **rspec-rails-1.3 (http://rspec.info)** .

Install

```
gem install rspec-rails
```

This installs the following gems:

```
rspec
rspec-core
rspec-expectations
rspec-mocks
rspec-rails
```

Configure

Add rspec-rails to the :test and :development groups in the Gemfile:

```
group :test, :development do
  gem "rspec-rails", "~> 2.4"
end
```

It needs to be in the :development group to expose generators and rake tasks without having to type RAILS_ENV=test.

Now you can run:

```
script/rails generate rspec:install
```

This adds the spec directory and some skeleton files, including a .rspec file.

Webrat and Capybara

You can choose between webrat or capybara for simulating a browser, automating a browser, or setting expectations using the matchers they supply. Just add your preference to the Gemfile:

```
gem "webrat"
gem "capybara"
```

Issues

The documentation for rspec-rails is a work in progress. We'll be adding Cucumber features over time, and clarifying existing ones. If you have specific features you'd like to see added, find the existing documentation incomplete or confusing, or, better yet, wish to write a missing Cucumber feature yourself, please **submit an issue (http://github.com/rspec/rspec-rails/issues)** or a **pull request (http://github.com/rspec/rspec-rails)** .

In RSpec Rails 2.8:

- **Start from scratch (gettingstarted)**
- **Generators (generators)**
- **Transactions (transactions)**
- **Autotest (autotest)**
- **Changelog (changelog)**
- **Upgrade (upgrade)**
- **Rails versions (railsversions)**
- **Request Specs (request-specs)**
- **Model Specs (model-specs)**
- **Controller Specs (controller-specs)**
- **Helper Specs (helper-specs)**
- **Mailer Specs (mailer-specs)**
- **Routing Specs (routing-specs)**
- **View Specs (view-specs)**
- **Matchers (matchers)**
- **Mocks (mocks)**

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Documentation for
RSpec Rails 2.8

Start from scratch

Install Rails-3

```
$ gem install rails -v "~> 3.0.0"
```

Generate an app

```
$ rails new example
$ cd example
```

Add rspec-rails to the Gemfile

```
$ echo 'gem "rspec-rails", :group => [:development, :test]' >> Gemfile
```

Install the bundle

```
$ bundle install
```

Bootstrap RSpec

```
$ rails generate rspec:install
```

Generate a scaffold

```
$ rails generate scaffold Widgets name:string
```

This generates files in the `app` and `spec` directories. The files in the `app` directory are generated by Rails, and Rails delegates the generation of the files in the `spec` directory to RSpec.

Run migrations

```
$ rake db:migrate && rake db:test:prepare
```

Run RSpec

```
$ rake spec
```

or

```
$ rspec spec --format documentation
```

If all went well, you should see output ending with:

```
29 examples, 0 failures, 2 pending
```

This output also includes the following controller spec:

```
WidgetsController
  GET index
    assigns all widgets as @widgets
  GET show
    assigns the requested widget as @widget
  GET new
    assigns a new widget as @widget
  GET edit
    assigns the requested widget as @widget
  POST create
    with valid params
      creates a new Widget
      assigns a newly created widget as @widget
      redirects to the created widget
    with invalid params
      assigns a newly created but unsaved widget as @widget
      re-renders the 'new' template
  PUT update
    with valid params
      updates the requested widget
      assigns the requested widget as @widget
      redirects to the widget
    with invalid params
      assigns the widget as @widget
      re-renders the 'edit' template
  DELETE destroy
    destroys the requested widget
    redirects to the widgets list
```

Output like this can help to quickly gain a high level understanding of how an object behaves. It also exposes which cases have been specified and which have not. Note the balance between the examples for the `create` and `update` actions. If the `redirects to the widget` example was missing from one or the other, it would be easy to spot.

Take a look at the generated `spec/controllers/widgets_controller_spec.rb` to get a sense of how to organize your specs to generate output like this.

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Generators

If you type script/rails generate, the only RSpec generator you'll actually see is rspec:install. That's because RSpec is registered with Rails as the test framework, so whenever you generate application components like models, controllers, etc, RSpec specs are generated instead of Test::Unit tests.

Note that the generators are there to help you get started, but they are no substitute for writing your own examples, and they are only guaranteed to work out of the box for with Rails' defaults (ActiveRecord, no Capybara or Webrat).

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Transactions

When you run `rails generate rspec:install`, the `spec/spec_helper.rb` file includes the following configuration:

```
RSpec.configure do |config|
  config.use_transactional_fixtures = true
end
```

The name of this setting is a bit misleading. What it really means in Rails is "run every test method within a transaction." In the context of rspec-rails, it means "run every example within a transaction."

The idea is to start each example with a clean database, create whatever data is necessary for that example, and then remove that data by simply rolling back the transaction at the end of the example.

Disabling transactions

If you prefer to manage the data yourself, or using another tool like **database_cleaner (https://github.com/bmabey/database_cleaner)** to do it for you, simply tell RSpec to tell Rails not to manage transactions:

```
RSpec.configure do |config|
  config.use_transactional_fixtures = false
end
```

Data created in `before(:each)` are rolled back

Any data you create in a `before(:each)` hook will be rolled back at the end of the example. This is a good thing because it means that each example is isolated from state that would otherwise be left around by the examples that already ran. For example:

```
describe Widget do
  before(:each) do
    @widget = Widget.create
  end

  it "does something" do
    @widget.should do_something
  end

  it "does something else" do
    @widget.should do_something_else
  end
end
```

The `@widget` is recreated in each of the two examples above, so each example has a different object, *and* the underlying data is rolled back so the data backing the `@widget` in each example is new.

Data created in `before(:all)` are *not* rolled back

`before(:all)` hooks are invoked before the transaction is opened. You can use this to speed things up by creating data once before any example in a group is run, however, this introduces a number of complications

and you should only do this if you have a firm grasp of the implications. Here are a couple of guidelines:

1. Be sure to clean up any data in an after(:all) hook:

   ```
   before(:all) do
     @widget = Widget.create!
   end

   after(:all) do
     @widget.destroy
   end
   ```

   If you don't do that, you'll leave data lying around that will eventually interfere with other examples.

2. Reload the object in a before(:each) hook.

   ```
   before(:all) do
     @widget = Widget.create!
   end

   before(:each) do
     @widget.reload
   end
   ```

   Even though database updates in each example will be rolled back, the object won't *know* about those rollbacks so the object and its backing data can easily get out of sync.

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Autotest

The rspec:install generator creates a .rspec file, which tells RSpec to tell Autotest that you're using RSpec. You'll also need to add the ZenTest and autotest-rails gems to your Gemfile:

```
gem "ZenTest", "~> 4.4.2"
gem "autotest-rails", "~> 4.1.0"
```

If all of the gems in your Gemfile are installed in system gems, you can just type

```
autotest
```

If Bundler is managing any gems for you directly (i.e. you've got :git or :path attributes in the Gemfile), however, you may need to run

```
bundle exec autotest
```

If you do, you require Autotest's bundler plugin in a .autotest file in the project root directory or your home directory:

```
require "autotest/bundler"
```

Now you can just type autotest, it should prefix the generated shell command with bundle exec.

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Documentation for
RSpec Rails 2.8

Rails versions

rails version	rspec-rails version
2.3	1.3.4
3.0	>= 2.0
3.1	>= 2.6

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

request spec

Request specs provide a thin wrapper around Rails' integration tests, and are designed to drive behavior through the full stack, including routing (provided by Rails) and without stubbing (that's up to you).

With request specs, you can:

- specify a single request
- specify multiple requests across multiple controllers
- specify multiple requests across multiple sessions

Check the rails documentation on integration tests for more information.

RSpec provides two matchers that delegate to Rails assertions:

```
render_template # delegates to assert_template
redirect_to     # delegates to assert_redirected_to
```

Check the Rails docs for details on these methods as well.

If you would like to use webrat or capybara with your request specs, all you have to do is include one of them in your Gemfile and RSpec will automatically load them in a request spec.

Scenario: specify managing a Widget with Rails integration methods (#0)

Given a file named "spec/requests/widget_management_spec.rb" with:

```
require "spec_helper"

describe "Widget management" do

  it "creates a Widget and redirects to the Widget's page" do
    get "/widgets/new"
    response.should render_template(:new)

    post "/widgets", :widget => {:name => "My Widget"}

    response.should redirect_to(assigns(:widget))
    follow_redirect!

    response.should render_template(:show)
    response.body.should include("Widget was successfully created.")
  end

end
```

When I run `rspec spec/requests/widget_management_spec.rb`

Then the example should pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Model Specs

Model specs live in spec/models or any example group with :type => :model.

A model spec is a thin wrapper for an ActiveSupport::TestCase, and includes all of the behavior and assertions that it provides, in addition to RSpec's own behavior and expectations.

Examples

```
require "spec_helper"

describe Post do
  context "with 2 or more comments" do
    it "orders them in reverse" do
      post = Post.create
      comment1 = post.comment("first")
      comment2 = post.comment("second")
      post.reload.comments.should eq([comment2, comment1])
    end
  end
end
```

In Model Specs:

- **errors_on (model-specs/errors-on)**
- **transactional examples (model-specs/transactional-examples)**

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

errors_on

Scenario: with one validation error (#0)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

class ValidatingWidget < ActiveRecord::Base
  set_table_name :widgets
  validates_presence_of :name
end

describe ValidatingWidget do
  it "fails validation with no name (using error_on)" do
    ValidatingWidget.new.should have(1).error_on(:name)
  end

  it "fails validation with no name (using errors_on)" do
    ValidatingWidget.new.should have(1).errors_on(:name)
  end

  it "passes validation with a name (using 0)" do
    ValidatingWidget.new(:name => "liquid nitrogen").should have(0).errors_on(:name)
  end

  it "passes validation with a name (using :no)" do
    ValidatingWidget.new(:name => "liquid nitrogen").should have(:no).errors_on(:name)
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

transactional examples

By default rspec executes each individual example in a transaction.

You can also explicitly enable/disable transactions the configuration property 'use*transactional*examples'.

Scenario: run in transactions (default) (#0)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe Widget do
  it "has none to begin with" do
    Widget.count.should == 0
  end

  it "has one after adding one" do
    Widget.create
    Widget.count.should == 1
  end

  it "has none after one was created in a previous example" do
    Widget.count.should == 0
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: run in transactions (explicit) (#1)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

RSpec.configure do |c|
  c.use_transactional_examples = true
end

describe Widget do
  it "has none to begin with" do
    Widget.count.should == 0
  end

  it "has one after adding one" do
    Widget.create
    Widget.count.should == 1
  end
```

```
  it "has none after one was created in a previous example" do
    Widget.count.should == 0
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: disable transactions (explicit) (#2)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

RSpec.configure do |c|
  c.use_transactional_examples = false
end

describe Widget do
  it "has none to begin with" do
    Widget.count.should == 0
  end

  it "has one after adding one" do
    Widget.create
    Widget.count.should == 1
  end

  it "has one after one was created in a previous example" do
    Widget.count.should == 1
  end

  after(:all) { Widget.destroy_all }
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: run in transactions with fixture (#3)

Given a file named "spec/models/thing_spec.rb" with:

```
require "spec_helper"

describe Thing do
  fixtures :things
  it "fixture method defined" do
    things(:one)
  end
end
```

Given a file named "spec/fixtures/things.yml" with:

```
one:
  name: MyString
```

When I run `rspec spec/models/thing_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Controller Specs

Controller specs live in spec/controllers or any example group with :type => :controller.

A controller spec is an RSpec wrapper for a Rails functional test (**ActionController::TestCase::Behavior (https://github.com/rails/rails/blob/master/actionpack/lib/action_controller/test_case.rb)**). It allows you to simulate a single http request in each example, and then specify expected outcomes such as:

- rendered templates
- redirects
- instance variables assigned in the controller to be shared with the view
- cookies sent back with the response

To specify outcomes, you can use:

- standard rspec matchers (response.code.should eq(200))
- standard test/unit assertions (assert_equal 200, response.code)
- rails assertions (assert_response 200)
- rails-specific matchers:
  - response.should render_template (wraps assert_template)
  - response.should redirect_to (wraps assert_redirected_to)
  - assigns(:widget).should be_a_new(Widget)

Examples

```
describe TeamsController do
  describe "GET index" do
    it "assigns @teams" do
      team = Team.create
      get :index
      assigns(:teams).should eq([team])
    end

    it "renders the index template" do
      get :index
      response.should render_template("index")
    end
  end
end
```

Views

- by default, views are not rendered. See **views are stubbed by default (controller-specs/views-are-stubbed-by-default)** and **render_views (controller-specs/render-views)** for details.

In Controller Specs:

- **Cookies (controller-specs/cookies)**
- **controller spec (controller-specs/controller-spec)**

- **views are stubbed by default** (controller-specs/views-are-stubbed-by-default)
- **render_views** (controller-specs/render-views)
- **anonymous controller** (controller-specs/anonymous-controller)
- **bypass rescue** (controller-specs/bypass-rescue)

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Cookies

Controller specs wrap Rails controller tests, which expose a few different ways to access cookies:

```
@request.cookies['key']
@response.cookies['key']
cookies['key']
```

rails-3.0.x and 3.1 handle these slightly differently, so to avoid confusion, we recommend the following guidelines:

Recommended guidelines for rails-3.0.0 to 3.1.0

- Access cookies through the request and response objects in the spec.
  - Use request.cookies before the action to set up state.
  - Use response.cookies after the action to specify outcomes.
- Use the cookies object in the controller action.
- Use String keys.

```
# spec
request.cookies['foo'] = 'bar'
get :some_action
response.cookies['foo'].should eq('modified bar')

# controller
def some_action
  cookies['foo'] = "modified #{cookies['foo']}"
end
```

Why use Strings instead of Symbols?
The cookies objects in the spec come from Rack, and do not support indifferent access (i.e. :foo and "foo" are different keys). The cookies object in the controller *does* support indifferent access, which is a bit confusing.

This changed in rails-3.1, so you *can* use symbol keys, but we recommend sticking with string keys for consistency.

Why not use the cookies method?
The cookies method combines the request and response cookies. This can lead to confusion when setting cookies in the example in order to set up state for the controller action.

```
# does not work in rails 3.0.0 > 3.1.0
cookies['foo'] = 'bar' # this is not visible in the controller
get :some_action
```

Future versions of Rails
There is code in the master branch in rails that makes cookie access more consistent so you can use the same cookies object before and after the action, and you can use String or Symbol keys. We'll update these docs accordingly when that is released.

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

controller spec

Scenario: **simple passing example (#0)**

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  describe "GET index" do
    it "has a 200 status code" do
      get :index
      response.code.should eq("200")
    end
  end
end
```

When I run `rspec spec`

Then the example should pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

views are stubbed by default

By default, controller specs stub views with a template that renders an empty string instead of the views in the app. This allows you specify which view template an action should try to render regardless of whether the template compiles cleanly.

NOTE: unlike rspec-rails-1.x, the real template must exist.

Scenario: **expect template that is rendered by controller action (passes) (#0)**

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  describe "index" do
    it "renders the index template" do
      get :index
      response.should render_template("index")
      response.body.should == ""
    end
    it "renders the widgets/index template" do
      get :index
      response.should render_template("widgets/index")
      response.body.should == ""
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: **expect template that is not rendered by controller action (fails) (#1)**

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  describe "index" do
    it "renders the 'new' template" do
      get :index
      response.should render_template("new")
    end
  end
end
```

When I run `rspec spec`

Then the output should contain "1 example, 1 failure"

Scenario: **expect empty templates to render when view path is changed at runtime (passes) (#2)**

Given a file named "spec/controllers/things_controller_spec.rb" with:

```
require "spec_helper"

describe ThingsController do
  describe "custom_action" do
    it "renders an empty custom_action template" do
      controller.prepend_view_path 'app/views'
      controller.append_view_path 'app/views'
      get :custom_action
      response.should render_template("custom_action")
      response.body.should == ""
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: **expect template to render when view path is changed at runtime (fails) (#3)**

Given a file named "spec/controllers/things_controller_spec.rb" with:

```
require "spec_helper"

describe ThingsController do
  describe "custom_action" do
    it "renders the custom_action template" do
      render_views
      controller.prepend_view_path 'app/views'
      get :custom_action
      response.should render_template("custom_action")
      response.body.should == ""
    end

    it "renders an empty custom_action template" do
      controller.prepend_view_path 'app/views'
      get :custom_action
      response.should render_template("custom_action")
      response.body.should == ""
    end
  end
end
```

When I run `rspec spec`

Then the output should contain "2 examples, 1 failure"

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

render_views

You can tell a controller example group to render views with the render_views declaration in any individual group, or globally.

Scenario: render_views directly in a single group (#0)

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  render_views

  describe "GET index" do
    it "says 'Listing widgets'" do
      get :index
      response.body.should =~ /Listing widgets/m
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: render_views on and off in nested groups (#1)

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  context "with render_views" do
    render_views

    describe "GET index" do
      it "renders the actual template" do
        get :index
        response.body.should =~ /Listing widgets/m
      end
    end

    context "with render_views(false) nested in a group with render_views" do
      render_views false

      describe "GET index" do
        it "renders the RSpec generated template" do
          get :index
          response.body.should eq("")
        end
```

```
      end
    end
  end
end

context "without render_views" do
  describe "GET index" do
    it "renders the RSpec generated template" do
      get :index
      response.body.should eq("")
    end
  end
end

context "with render_views again" do
  render_views

  describe "GET index" do
    it "renders the actual template" do
      get :index
      response.body.should =~ /Listing widgets/m
    end
  end
end
end
```

When I run `rspec spec --format documentation`

Then the output should contain:

```
WidgetsController
  with render_views
    GET index
      renders the actual template
    with render_views(false) nested in a group with render_views
      GET index
        renders the RSpec generated template
  without render_views
    GET index
      renders the RSpec generated template
  with render_views again
    GET index
      renders the actual template
```

Scenario: render_views globally (#2)

Given a file named "spec/support/render_views.rb" with:

```
RSpec.configure do |config|
  config.render_views
end
```

And a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"
```

```
require 'spec_helper'

describe WidgetsController do
  describe "GET index" do
    it "renders the index template" do
      get :index
      response.body.should =~ /Listing widgets/m
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

anonymous controller

Use the controller method to define an anonymous controller derived from ApplicationController. This is useful for specifying behavior like global error handling.

To specify a different base class, you can pass the class explicitly to the controller method:

```
controller(BaseController)
```

You can also configure RSpec to use the described class:

```
RSpec.configure do |c|
  c.infer_base_class_for_anonymous_controllers = true
end

describe BaseController do
  controller { ... }
```

Scenario: specify error handling in ApplicationController (#0)

Given a file named "spec/controllers/application_controller_spec.rb" with:

```
require "spec_helper"

class ApplicationController < ActionController::Base
  class AccessDenied < StandardError; end

  rescue_from AccessDenied, :with => :access_denied

private

  def access_denied
    redirect_to "/401.html"
  end
end

describe ApplicationController do
  controller do
    def index
      raise ApplicationController::AccessDenied
    end
  end

  describe "handling AccessDenied exceptions" do
    it "redirects to the /401.html page" do
      get :index
      response.should redirect_to("/401.html")
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: specify error handling in subclass of ApplicationController (#1)

Given a file named "spec/controllers/application_controller_subclass_spec.rb" with:

```
require "spec_helper"

class ApplicationController < ActionController::Base
  class AccessDenied < StandardError; end
end

class ApplicationControllerSubclass < ApplicationController

  rescue_from ApplicationController::AccessDenied, :with => :access_denied

private

  def access_denied
    redirect_to "/401.html"
  end
end

describe ApplicationControllerSubclass do
  controller(ApplicationControllerSubclass) do
    def index
      raise ApplicationController::AccessDenied
    end
  end

  describe "handling AccessDenied exceptions" do
    it "redirects to the /401.html page" do
      get :index
      response.should redirect_to("/401.html")
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: infer base class from the described class (#2)

Given a file named "spec/controllers/base_class_can_be_inferred_spec.rb" with:

```
require "spec_helper"

RSpec.configure do |c|
  c.infer_base_class_for_anonymous_controllers = true
end

class ApplicationController < ActionController::Base; end
```

```
class ApplicationControllerSubclass < ApplicationController; end

describe ApplicationControllerSubclass do
  controller do
    def index
      render :text => "Hello World"
    end
  end

  it "creates an anonymous controller derived from ApplicationControllerSubclass" do
    controller.should be_a_kind_of(ApplicationControllerSubclass)
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: **invoke around filter in base class (#3)**

Given a file named "spec/controllers/application_controller_around_filter_spec.rb" with:

```
require "spec_helper"

class ApplicationController < ActionController::Base
  around_filter :an_around_filter

  def an_around_filter
    @callback_invoked = true
    yield
  end
end

describe ApplicationController do
  controller do
    def index
      render :nothing => true
    end
  end

  it "invokes the callback" do
    get :index

    assigns[:callback_invoked].should be_true
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: **anonymous controllers only create resource routes (#4)**

Given a file named "spec/controllers/application_controller_spec.rb" with:

```
require "spec_helper"

describe ApplicationController do
  controller do
    def index
      render :text => "index called"
    end

    def create
      render :text => "create called"
    end

    def new
      render :text => "new called"
    end

    def show
      render :text => "show called"
    end

    def edit
      render :text => "edit called"
    end

    def update
      render :text => "update called"
    end

    def destroy
      render :text => "destroy called"
    end

    def willerror
      render :text => "will not render"
    end
  end

  describe "#index" do
    it "responds to GET" do
      get :index
      response.body.should == "index called"
    end

    it "also responds to POST" do
      post :index
      response.body.should == "index called"
    end

    it "also responds to PUT" do
      put :index
      response.body.should == "index called"
    end

    it "also responds to DELETE" do
      delete :index
      response.body.should == "index called"
    end
  end

  describe "#create" do
    it "responds to POST" do
      post :create
      response.body.should == "create called"
```

```
  end

  # And the rest...
  %w{get post put delete}.each do |calltype|
    it "responds to #{calltype}" do
      send(calltype, :create)
      response.body.should == "create called"
    end
  end
end

describe "#new" do
  it "responds to GET" do
    get :new
    response.body.should == "new called"
  end

  # And the rest...
  %w{get post put delete}.each do |calltype|
    it "responds to #{calltype}" do
      send(calltype, :new)
      response.body.should == "new called"
    end
  end
end

describe "#edit" do
  it "responds to GET" do
    get :edit, :id => "anyid"
    response.body.should == "edit called"
  end

  it "requires the :id parameter" do
    expect { get :edit }.to raise_error(ActionController::RoutingError)
  end

  # And the rest...
  %w{get post put delete}.each do |calltype|
    it "responds to #{calltype}" do
      send(calltype, :edit, {:id => "anyid"})
      response.body.should == "edit called"
    end
  end
end

describe "#show" do
  it "responds to GET" do
    get :show, :id => "anyid"
    response.body.should == "show called"
  end

  it "requires the :id parameter" do
    expect { get :show }.to raise_error(ActionController::RoutingError)
  end

  # And the rest...
  %w{get post put delete}.each do |calltype|
    it "responds to #{calltype}" do
      send(calltype, :show, {:id => "anyid"})
      response.body.should == "show called"
    end
  end
end
```

```
  describe "#update" do
    it "responds to PUT" do
      put :update, :id => "anyid"
      response.body.should == "update called"
    end

    it "requires the :id parameter" do
      expect { put :update }.to raise_error(ActionController::RoutingError)
    end

    # And the rest...
    %w{get post put delete}.each do |calltype|
      it "responds to #{calltype}" do
        send(calltype, :update, {:id => "anyid"})
        response.body.should == "update called"
      end
    end
  end

  describe "#destroy" do
    it "responds to DELETE" do
      delete :destroy, :id => "anyid"
      response.body.should == "destroy called"
    end

    it "requires the :id parameter" do
      expect { delete :destroy }.to raise_error(ActionController::RoutingError)
    end

    # And the rest...
    %w{get post put delete}.each do |calltype|
      it "responds to #{calltype}" do
        send(calltype, :destroy, {:id => "anyid"})
        response.body.should == "destroy called"
      end
    end
  end

  describe "#willerror" do
    it "cannot be called" do
      expect { get :willerror }.to raise_error(ActionController::RoutingError)
    end
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

bypass rescue

Use bypass_rescue to bypass both Rails' default handling of errors in controller actions, and any custom handling declared with a rescue_from statement.

This lets you specify details of the exception being raised, regardless of how it might be handled upstream.

Background: (#0)

Given a file named "spec/controllers/gadgets_controller_spec_context.rb" with:

```
class AccessDenied < StandardError; end

class ApplicationController < ActionController::Base
  rescue_from AccessDenied, :with => :access_denied

  private

  def access_denied
    redirect_to "/401.html"
  end
end
```

Scenario: standard exception handling using `rescue_from` (#1)

Given a file named "spec/controllers/gadgets_controller_spec.rb" with:

```
require "spec_helper"

require 'controllers/gadgets_controller_spec_context'

describe GadgetsController do
  before do
    def controller.index
      raise AccessDenied
    end
  end

  describe "index" do
    it "redirects to the /401.html page" do
      get :index
      response.should redirect_to("/401.html")
    end
  end
end
```

When I run `rspec spec/controllers/gadgets_controller_spec.rb`

Then the examples should all pass

Then the examples should all pass

Scenario: bypass `rescue_from` handling with `bypass_rescue` (#2)

Given a file named "spec/controllers/gadgets_controller_spec.rb" with:

```
require "spec_helper"

require 'controllers/gadgets_controller_spec_context'

describe GadgetsController do
  before do
    def controller.index
      raise AccessDenied
    end
  end

  describe "index" do
    it "raises AccessDenied" do
      bypass_rescue
      expect { get :index }.to raise_error(AccessDenied)
    end
  end
end
```

When I run `rspec spec/controllers/gadgets_controller_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

helper spec

Helper specs live in spec/helpers, or any example group with :type => :helper.

Helper specs expose a helper object, which includes the helper module being specified, the ApplicationHelper module (if there is one) and all of the helpers built into Rails. It does not include the other helper modules in your app.

To access the helper methods you're specifying, simply call them directly on the helper object.

NOTE: helper methods defined in controllers are not included.

Scenario: helper method that returns a value (#0)

Given a file named "spec/helpers/application_helper_spec.rb" with:

```
require "spec_helper"

describe ApplicationHelper do
  describe "#page_title" do
    it "returns the default title" do
      helper.page_title.should eq("RSpec is your friend")
    end
  end
end
```

And a file named "app/helpers/application_helper.rb" with:

```
module ApplicationHelper
  def page_title
    "RSpec is your friend"
  end
end
```

When I run `rspec spec/helpers/application_helper_spec.rb`

Then the examples should all pass

Scenario: helper method that accesses an instance variable (#1)

Given a file named "spec/helpers/application_helper_spec.rb" with:

```
require "spec_helper"

describe ApplicationHelper do
  describe "#page_title" do
    it "returns the instance variable" do
```

```
    it "returns the instance variable" do
      assign(:title, "My Title")
      helper.page_title.should eql("My Title")
    end
  end
end
```

And a file named "app/helpers/application_helper.rb" with:

```
module ApplicationHelper
  def page_title
    @title || nil
  end
end
```

When I run `rspec spec/helpers/application_helper_spec.rb`

Then the examples should all pass

Scenario: application helper is included in helper object (#2)

Given a file named "spec/helpers/widgets_helper_spec.rb" with:

```
require "spec_helper"

describe WidgetsHelper do
  describe "#page_title" do
    it "includes the app name" do
      assign(:title, "This Page")
      helper.page_title.should eq("The App: This Page")
    end
  end
end
```

And a file named "app/helpers/application_helper.rb" with:

```
module ApplicationHelper
  def app_name
    "The App"
  end
end
```

And a file named "app/helpers/widgets_helper.rb" with:

```
module WidgetsHelper
  def page_title
    "#{app_name}: #{@title}"
  end
end
```

When I run `rspec spec/helpers/widgets_helper_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

URL helpers in mailer examples

Scenario: using URL helpers with default options (#0)

Given a file named "config/initializers/mailer_defaults.rb" with:

```
Rails.configuration.action_mailer.default_url_options = { :host => 'example.com' }
```

And a file named "spec/mailers/notifications_spec.rb" with:

```
require 'spec_helper'

describe Notifications do
  it 'should have access to URL helpers' do
    lambda { gadgets_url }.should_not raise_error
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Scenario: using URL helpers without default options (#1)

Given a file named "config/initializers/mailer_defaults.rb" with:

```
# no default options
```

And a file named "spec/mailers/notifications_spec.rb" with:

```
require 'spec_helper'

describe Notifications do
  it 'should have access to URL helpers' do
    lambda { gadgets_url :host => 'example.com' }.should_not raise_error
    lambda { gadgets_url }.should raise_error
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Routing Specs

Routing specs live in the spec/routing directory, or any example group with :type => :routing.

Simple apps with nothing but standard RESTful routes won't get much value from routing specs, but they can provide significant value when used to specify customized routes, like vanity links, slugs, etc.

```
{ :get => "/articles/2012/11/when-to-use-routing-specs" }.
  should route_to(
    :controller => "articles",
    :month => "2012-11",
    :slug => "when-to-use-routing-specs"
  )
```

They are also valuable for routes that should not be available:

```
{ :delete => "/accounts/37" }.should_not be_routable
```

In Routing Specs:

- **route_to matcher (routing-specs/route-to-matcher)**
- **be_routable matcher (routing-specs/be-routable-matcher)**
- **named routes (routing-specs/named-routes)**

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

route_to matcher

The route_to matcher specifies that a request (verb + path) is routable. It is most valuable when specifying routes other than standard RESTful routes.

```
get("/").should route_to("welcome#index") # new in 2.6.0

or

{ :get => "/" }.should route_to(:controller => "welcome")
```

Scenario: passing route spec with shortcut syntax (#0)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "routes /widgets to the widgets controller" do
    get("/widgets").
      should route_to("widgets#index")
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the examples should all pass

Scenario: passing route spec with verbose syntax (#1)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "routes /widgets to the widgets controller" do
    { :get => "/widgets" }.
      should route_to(:controller => "widgets", :action => "index")
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the examples should all pass

Scenario: route spec for a route that doesn't exist (fails) (#2)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "routes /widgets/foo to the /foo action" do
    get("/widgets/foo").should route_to("widgets#foo")
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the output should contain "1 failure"

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

be_routable matcher

The be_routable matcher is best used with should_not to specify that a given route should not be routable. It is available in routing specs (in spec/routing) and controller specs (in spec/controllers).

Scenario: specify routeable route should not be routable (fails) (#0)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "does not route to widgets" do
    { :get => "/widgets" }.should_not be_routable
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the output should contain "1 example, 1 failure"

Scenario: specify non-routeable route should not be routable (passes) (#1)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "does not route to widgets/foo/bar" do
    { :get => "/widgets/foo/bar" }.should_not be_routable
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the examples should all pass

Scenario: specify routeable route should be routable (passes) (#2)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "routes to /widgets" do
    { :get => "/widgets" }.should be_routable
  end
```

```
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the examples should all pass

Scenario: specify non-routeable route should be routable (fails) (#3)

Given a file named "spec/routing/widgets_routing_spec.rb" with:

```
require "spec_helper"

describe "routes for Widgets" do
  it "routes to widgets/foo/bar" do
    { :get => "/widgets/foo/bar" }.should be_routable
  end
end
```

When I run `rspec spec/routing/widgets_routing_spec.rb`

Then the output should contain "1 example, 1 failure"

Scenario: be_routable in a controller spec (#4)

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do
  it "routes to /widgets" do
    { :get => "/widgets" }.should be_routable
  end
end
```

When I run `rspec spec/controllers/widgets_controller_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

named routes

Routing specs have access to named routes.

Scenario: access named route (#0)

Given a file named "spec/routing/widget_routes_spec.rb" with:

```
require "spec_helper"

describe "routes to the widgets controller" do
  it "routes a named route" do
    {:get => new_widget_path}.
      should route_to(:controller => "widgets", :action => "new")
  end
end
```

When I run `rspec spec`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

view spec

View specs live in spec/views and render view templates in isolation.

Scenario: passing spec that renders the described view file (#0)

Given a file named "spec/views/widgets/index.html.erb_spec.rb" with:

```
require "spec_helper"

describe "widgets/index.html.erb" do
  it "displays all the widgets" do
    assign(:widgets, [
      stub_model(Widget, :name => "slicer"),
      stub_model(Widget, :name => "dicer")
    ])

    render

    rendered.should =~ /slicer/
    rendered.should =~ /dicer/
  end
end
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing spec with before and nesting (#1)

Given a file named "spec/views/widgets/index.html.erb_spec.rb" with:

```
require "spec_helper"

describe "widgets/index.html.erb" do

  context "with 2 widgets" do
    before(:each) do
      assign(:widgets, [
        stub_model(Widget, :name => "slicer"),
        stub_model(Widget, :name => "dicer")
      ])
    end

    it "displays both widgets" do
      render

      rendered.should =~ /slicer/
      rendered.should =~ /dicer/
    end
  end
```

```
end
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing spec with explicit template rendering (#2)

Given a file named "spec/views/widgets/widget.html.erb_spec.rb" with:

```
require "spec_helper"

describe "rendering the widget template" do
  it "displays the widget" do
    assign(:widget, stub_model(Widget, :name => "slicer"))

    render :template => "widgets/widget.html.erb"

    rendered.should =~ /slicer/
  end
end
```

And a file named "app/views/widgets/widget.html.erb" with:

```
<h2><%= @widget.name %></h2>
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing spec with rendering of locals in a partial (#3)

Given a file named "spec/views/widgets/_widget.html.erb_spec.rb" with:

```
require "spec_helper"

describe "rendering locals in a partial" do
  it "displays the widget" do
    widget = stub_model(Widget, :name => "slicer")

    render :partial => "widgets/widget.html.erb", :locals => {:widget => widget}

    rendered.should =~ /slicer/
  end
end
```

And a file named "app/views/widgets/_widget.html.erb" with:

```
<h3><%= widget.name %></h3>
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing spec with rendering of locals in an implicit partial (#4)

Given a file named "spec/views/widgets/_widget.html.erb_spec.rb" with:

```
require "spec_helper"

describe "rendering locals in a partial" do
  it "displays the widget" do
    widget = stub_model(Widget, :name => "slicer")

    render "widgets/widget", :widget => widget

    rendered.should =~ /slicer/
  end
end
```

And a file named "app/views/widgets/_widget.html.erb" with:

```
<h3><%= widget.name %></h3>
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing spec with rendering of text (#5)

Given a file named "spec/views/widgets/direct.html.erb_spec.rb" with:

```
require "spec_helper"

describe "rendering text directly" do
  it "displays the given text" do

    render :text => "This is directly rendered"

    rendered.should =~ /directly rendered/
  end
end
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: passing view spec that stubs a helper method (#6)

Given a file named "app/views/secrets/index.html.erb" with:

Given a file named "app/views/secrets/index.html.erb" with:

```
<%- if admin? %>
  <h1>Secret admin area</h1>
<%- end %>
```

And a file named "spec/views/secrets/index.html.erb_spec.rb" with:

```
require 'spec_helper'

describe 'secrets/index.html.erb' do
  before do
    view.stub(:admin?).and_return(true)
  end

  it 'checks for admin access' do
    render
    rendered.should =~ /Secret admin area/
  end
end
```

When I run `rspec spec/views/secrets`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

stub template

In order to isolate view specs from the partials rendered by the primary view, rspec-rails (since 2.2) provides the stub_template method.

Scenario: stub template that does not exist (#0)

Given a file named "spec/views/gadgets/list.html.erb_spec.rb" with:

```
require "spec_helper"

describe "gadgets/list.html.erb" do
  it "renders the gadget partial for each gadget" do
    assign(:gadgets, [
      mock_model(Gadget, :id => 1, :name => "First"),
      mock_model(Gadget, :id => 2, :name => "Second")
    ])
    stub_template "gadgets/_gadget.html.erb" => "<%= gadget.name %><br/>"
    render
    rendered.should =~ /First/
    rendered.should =~ /Second/
  end
end
```

And a file named "app/views/gadgets/list.html.erb" with:

```
<%= render :partial => "gadget", :collection => @gadgets %>
```

When I run `rspec spec/views/gadgets/list.html.erb_spec.rb`

Then the examples should all pass

Scenario: stub template that exists (#1)

Given a file named "spec/views/gadgets/edit.html.erb_spec.rb" with:

```
require "spec_helper"

describe "gadgets/edit.html.erb" do
  before(:each) do
    @gadget = assign(:gadget, stub_model(Gadget))
  end

  it "renders the form partial" do
    stub_template "gadgets/_form.html.erb" => "This content"
    render
    rendered.should =~ /This content/
  end
```

```
end
```

When I run `rspec spec/views/gadgets/edit.html.erb_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

view spec infers controller path and action

Scenario: infer controller path (#0)

Given a file named "spec/views/widgets/new.html.erb_spec.rb" with:

```
require "spec_helper"

describe "widgets/new.html.erb" do
  it "infers the controller path" do
    controller.request.path_parameters["controller"].should eq("widgets")
  end
end
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: infer action (#1)

Given a file named "spec/views/widgets/new.html.erb_spec.rb" with:

```
require "spec_helper"

describe "widgets/new.html.erb" do
  it "infers the controller path" do
    controller.request.path_parameters["action"].should eq("new")
  end
end
```

When I run `rspec spec/views`

Then the examples should all pass

Scenario: do not infer action in a partial (#2)

Given a file named "spec/views/widgets/_form.html.erb_spec.rb" with:

```
require "spec_helper"

describe "widgets/_form.html.erb" do
  it "includes a link to new" do
    controller.request.path_parameters["action"].should be_nil
  end
end
```

When I run `rspec spec/views`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

Matchers

rspec-rails offers a number of custom matchers, most of which are rspec-compatible wrappers for Rails' assertions.

redirects

```
# delegates to assert_redirected_to
response.should redirect_to(path)
```

templates

```
# delegates to assert_template
response.should render_template(template_name)
```

assigned objects

```
# passes if assigns(:widget) is an instance of Widget
# and it is not persisted
assigns(:widget).should be_a_new(Widget)
```

In Matchers:

- **be_a_new matcher (matchers/be-a-new-matcher)**
- **render_template matcher (matchers/render-template-matcher)**
- **redirect_to matcher (matchers/redirect-to-matcher)**
- **ActiveRecord::Relation match array (matchers/activerecord-relation-match-array)**

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

be_a_new matcher

The be_a_new matcher accepts a class and passes if the subject is an instance of that class that returns false to persisted?

You can also chain with on be_a_new with a hash of attributes to specify the subject has equal attributes.

Scenario: example spec with four be_a_new possibilities (#0)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe Widget do
  context "when initialized" do
    subject { Widget.new }
    it { should be_a_new(Widget) }
    it { should_not be_a_new(String) }
  end
  context "when saved" do
    subject { Widget.create }
    it { should_not be_a_new(Widget) }
    it { should_not be_a_new(String) }
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

render_template matcher

The render_template matcher is used to specify that a request renders a given template. It delegates to assert_template **(http://api.rubyonrails.org/classes/ActionController/TemplateAssertions.html#method-i-assert_template)**

It is available in controller specs (spec/controllers) and request specs (spec/requests).

NOTE: use redirect_to(:action => 'new') for redirects, not render_template.

Scenario: render_template with three possible options (#0)

Given a file named "spec/controllers/gadgets_spec.rb" with:

```
require "spec_helper"

describe GadgetsController do
  describe "GET #index" do
    subject { get :index }

    it { should render_template(:index) }
    it { should render_template("index") }
    it { should render_template("gadgets/index") }
  end
end
```

When I run `rspec spec/controllers/gadgets_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

redirect_to matcher

The redirect_to matcher is used to specify that a request redirects to a given template or action. It delegates to assert_redirected_to **(http://api.rubyonrails.org/classes/ActionDispatch/Assertions/ResponseAssertions.html#method-i-assert_redirected_to)** .

It is available in controller specs (spec/controllers) and request specs (spec/requests).

Scenario: **redirect_to with four possible options (#0)**

Given a file named "spec/controllers/widgets_controller_spec.rb" with:

```
require "spec_helper"

describe WidgetsController do

  describe "#create" do
    subject { post :create, :widget => { :name => "Foo" } }

    it "redirects to widget_url(@widget)" do
      subject.should redirect_to(widget_url(assigns(:widget)))
    end

    it "redirects_to :action => :show" do
      subject.should redirect_to :action => :show,
                                 :id => assigns(:widget).id
    end

    it "redirects_to(@widget)" do
      subject.should redirect_to(assigns(:widget))
    end

    it "redirects_to /widgets/:id" do
      subject.should redirect_to("/widgets/#{assigns(:widget).id}")
    end
  end
end
```

When I run `rspec spec/controllers/widgets_controller_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

ActiveRecord::Relation match array

The =~ matcher can be used with an ActiveRecord::Relation (scope). The assertion will pass if the scope would return all of the elements specified in the array on the right hand side.

Scenario: example spec with relation =~ matcher (#0)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe Widget do
  let!(:widgets) { Array.new(3) { Widget.create } }
  subject { Widget.scoped }

  it { should =~ widgets }
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 Matt Wynne Ltd. We claim no intellectual property rights over the material provided to this service.
Printed from www.relish.com

mock_model

The mock_model method generates a test double that acts like an instance of ActiveModel. This is different from the stub_model method which generates an instance of a real model class.

The benefit of mock_model over stub_model is that it is a true double, so the examples are not dependent on the behavior (or mis-behavior), or even the existence of any other code. If you're working on a controller spec and you need a model that doesn't exist, you can pass mock_model a string and the generated object will act as though its an instance of the class named by that string.

Scenario: passing a string that represents a non-existent constant (#0)

Given a file named "spec/models/car_spec.rb" with:

```
require "spec_helper"

describe "mock_model('Car') with no Car constant in existence" do
  it "generates a constant" do
    Object.const_defined?(:Car).should be_false
    mock_model("Car")
    Object.const_defined?(:Car).should be_true
  end

  describe "generates an object that ..." do
    it "returns the correct name" do
      car = mock_model("Car")
      car.class.name.should eq("Car")
    end

    it "says it is a Car" do
      car = mock_model("Car")
      car.should be_a(Car)
    end
  end
end
```

When I run `rspec spec/models/car_spec.rb`

Then the examples should all pass

Scenario: passing a string that represents an existing constant (#1)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe Widget do
  it "uses the existing constant" do
    widget = mock_model("Widget")
    widget.should be_a(Widget)
```

```
    widget.should be_a(Widget)
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: passing a class that does not extend ActiveModel::Naming (#2)

Given a file named "spec/models/string_spec.rb" with:

```
require "spec_helper"

describe String do
  it "raises" do
    expect { mock_model(String) }.to raise_exception
  end
end
```

When I run `rspec spec/models/string_spec.rb`

Then the examples should all pass

Scenario: passing an Active Record constant (#3)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe Widget do
  let(:widget) { mock_model(Widget) }

  it "is valid by default" do
    widget.should be_valid
  end

  it "is not a new record by default" do
    widget.should_not be_new_record
  end

  it "can be converted to a new record" do
    widget.as_new_record.should be_new_record
  end

  it "sets :id to nil upon destroy" do
    widget.destroy
    widget.id.should be_nil
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: passing an Active Record constant with method stubs (#4)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe "mock_model(Widget) with stubs" do
  let(:widget) do
    mock_model Widget, :foo => "bar",
                       :save => true,
                       :update_attributes => false
  end

  it "supports stubs for methods that don't exist in ActiveModel or ActiveRecord" do
    widget.foo.should eq("bar")
  end

  it "supports stubs for methods that do exist" do
    widget.save.should eq(true)
    widget.update_attributes.should be_false
  end

  describe "#errors" do
    context "with update_attributes => false" do
      it "is not empty" do
        widget.errors.should_not be_empty
      end
    end
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: mock_model outside rails (#5)

Given a file named "mock_model_outside_rails_spec.rb" with:

```
require 'rspec/rails/mocks'

describe "Foo" do
  it "is mockable" do
    foo = mock_model("Foo")
    foo.id.should eq(1001)
    foo.to_param.should eq("1001")
  end
end
```

When I run `rspec mock_model_outside_rails_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky

Last updated 3 months ago by dcheilmsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**

stub_model

The stub_model method generates an instance of a Active Model model.

While you can use stub_model in any example (model, view, controller, helper), it is especially useful in view examples, which are inherently more state-based than interaction-based.

Scenario: passing an Active Record constant with a hash of stubs (#0)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe "stub_model(Widget) with a hash of stubs" do
  let(:widget) do
    stub_model Widget, :id => 5, :random_attribute => true
  end

  it "stubs :id" do
    widget.id.should eql(5)
  end

  it "stubs :random_attribute" do
    widget.random_attribute.should be_true
  end

  it "returns false for new_record? if :id is set" do
    widget.should_not be_new_record
  end

  it "can be converted to a new record" do
    widget.as_new_record
    widget.should be_new_record
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Scenario: passing an Active Record constant with a block of stubs (#1)

Given a file named "spec/models/widget_spec.rb" with:

```
require "spec_helper"

describe "stub_model(Widget) with a block of stubs" do
  let(:widget) do
    stub_model Widget do |widget|
      widget.id = 5
```

```
    end
  end

  it "stubs :id" do
    widget.id.should eql(5)
  end
end
```

When I run `rspec spec/models/widget_spec.rb`

Then the examples should all pass

Last updated 3 months ago by dchelimsky.

©2012 **Matt Wynne Ltd**. We claim no intellectual property rights over the material provided to this service.
Printed from **www.relish.com**