Rails Application Templates

Application templates are simple Ruby files containing DSL for adding gems, initializers, etc. to your freshly created Rails project or an existing Rails project.

After reading this guide, you will know:

- How to use templates to generate/customize Rails applications.
- How to write your own reusable application templates using the Rails template API.

Usage

To apply a template, you need to provide the Rails generator with the location of the template you wish to apply using the `-m` option. This can either be a path to a file or a URL.

```
$ rails new blog -m ~/template.rb
$ rails new blog -m http://example.com/template.rb
```

You can use the `app:template` rails command to apply templates to an existing Rails application. The location of the template needs to be passed in via the LOCATION environment variable. Again, this can either be path to a file or a URL.

```
$ bin/rails app:template LOCATION=~/template.rb
$ bin/rails app:template LOCATION=http://example.com/template.rb
```

Template API

The Rails templates API is easy to understand. Here's an example of a typical Rails template:

```
# template.rb
generate(:scaffold, "person name:string")
route "root to: 'people#index'"
rails_command("db:migrate")

after_bundle do
  git :init
  git add: "."
  git commit: %Q{ -m 'Initial commit' }
end
```

The following sections outline the primary methods provided by the API:

gem(*args)

Adds a `gem` entry for the supplied gem to the generated application's `Gemfile`.

For example, if your application depends on the gems `bj` and `nokogiri`:

```ruby
gem "bj"
gem "nokogiri"
```

Please note that this will NOT install the gems for you and you will have to run `bundle install` to do that.

```
$ bundle install
```

gem_group(*names, &block)

Wraps gem entries inside a group.

For example, if you want to load `rspec-rails` only in the `development` and `test` groups:

```ruby
gem_group :development, :test do
  gem "rspec-rails"
end
```

add_source(source, options={}, &block)

Adds the given source to the generated application's `Gemfile`.

For example, if you need to source a gem from `"http://gems.github.com"`:

```ruby
add_source "http://gems.github.com"
```

If block is given, gem entries in block are wrapped into the source group.

```ruby
add_source "http://gems.github.com/" do
  gem "rspec-rails"
end
```

environment/application(data=nil, options={}, &block)

Adds a line inside the `Application` class for `config/application.rb`.

If `options[:env]` is specified, the line is appended to the corresponding file in `config/environments`.

```ruby
environment 'config.action_mailer.default_url_options = {host: "http://yourwebsite.example.
```

A block can be used in place of the `data` argument.

vendor/lib/file/initializer(filename, data = nil, &block)

Adds an initializer to the generated application's `config/initializers` directory.

Let's say you like using `Object#not_nil?` and `Object#not_blank?`:

```ruby
initializer 'bloatlol.rb', <<-CODE
  class Object
    def not_nil?
      !nil?
    end

    def not_blank?
      !blank?
    end
  end
CODE
```

Similarly, `lib()` creates a file in the `lib/` directory and `vendor()` creates a file in the `vendor/` directory.

There is even `file()`, which accepts a relative path from `Rails.root` and creates all the directories/files needed:

```ruby
file 'app/components/foo.rb', <<-CODE
  class Foo
  end
CODE
```

That'll create the `app/components` directory and put `foo.rb` in there.

rakefile(filename, data = nil, &block)

Creates a new rake file under `lib/tasks` with the supplied tasks:

```ruby
rakefile("bootstrap.rake") do
  <<-TASK
    namespace :boot do
      task :strap do
        puts "i like boots!"
      end
    end
  TASK
end
```

The above creates `lib/tasks/bootstrap.rake` with a `boot:strap` rake task.

generate(what, *args)

Runs the supplied rails generator with given arguments.

```
generate(:scaffold, "person", "name:string", "address:text", "age:number")
```

run(command)

Executes an arbitrary command. Just like the backticks. Let's say you want to remove the `README.rdoc` file:

```
run "rm README.rdoc"
```

rails_command(command, options = {})

Runs the supplied command in the Rails application. Let's say you want to migrate the database:

```
rails_command "db:migrate"
```

You can also run commands with a different Rails environment:

```
rails_command "db:migrate", env: 'production'
```

You can also run commands as a super-user:

```
rails_command "log:clear", sudo: true
```

You can also run commands that should abort application generation if they fail:

```
rails_command "db:migrate", abort_on_failure: true
```

route(routing_code)

Adds a routing entry to the `config/routes.rb` file. In the steps above, we generated a person scaffold and also removed `README.rdoc`. Now, to make `PeopleController#index` the default page for the application:

```
route "root to: 'person#index'"
```

inside(dir)

Enables you to run a command from the given directory. For example, if you have a copy of edge rails that you wish to symlink from your new apps, you can do this:

```
inside('vendor') do
  run "ln -s ~/commit-rails/rails rails"
end
```

ask(question)

`ask()` gives you a chance to get some feedback from the user and use it in your templates. Let's say you want your user to name the new shiny library you're adding:

```
lib_name = ask("What do you want to call the shiny library ?")
lib_name << ".rb" unless lib_name.index(".rb")

lib lib_name, <<-CODE
  class Shiny
  end
CODE
```

yes?(question) or no?(question)

These methods let you ask questions from templates and decide the flow based on the user's answer. Let's say you want to prompt the user to run migrations:

```
rails_command("db:migrate") if yes?("Run database migrations?")
# no?(question) acts just the opposite.
```

git(:command)

Rails templates let you run any git command:

```
git :init
git add: "."
git commit: "-a -m 'Initial commit'"
```

after_bundle(&block)

Registers a callback to be executed after the gems are bundled and binstubs are generated. Useful for adding generated files to version control:

```
after_bundle do
  git :init
  git add: '.'
  git commit: "-a -m 'Initial commit'"
end
```

The callbacks gets executed even if `--skip-bundle` has been passed.

Advanced Usage

The application template is evaluated in the context of a `Rails::Generators::AppGenerator` instance. It uses the `apply` action provided by Thor.

This means you can extend and change the instance to match your needs.

For example by overwriting the `source_paths` method to contain the location of your template. Now methods like `copy_file` will accept relative paths to your template's location.

```ruby
def source_paths
  [__dir__]
end
```