```
demo> rails generate controller Say hello goodbye
create  app/controllers/say_controller.rb
 route  get "say/goodbye"
 route  get "say/hello"
invoke  erb
create    app/views/say
create    app/views/say/hello.html.erb
create    app/views/say/goodbye.html.erb
invoke  test_unit
create    test/controllers/say_controller_test.rb
invoke  helper
create    app/helpers/say_helper.rb
invoke    test_unit
create      test/helpers/say_helper_test.rb
invoke  assets
invoke    coffee
create      app/assets/javascripts/say.js.coffee
invoke    scss
create      app/assets/stylesheets/say.css.scss
```

The rails generate command logs the files and directories it examines, noting when it adds new Ruby scripts or directories to your application. For now, we're interested in one of these scripts and (in a minute) the .html.erb files.

The first source file we'll be looking at is the controller. You'll find it in the file app/controllers/say_controller.rb. Let's take a look at it:

Download rails40/demo1/app/controllers/say_controller.rb

```
class SayController < ApplicationController
  def hello
  end

  def goodbye
  end
end
```

defining classes
↪ on page 45

Pretty minimal, eh? SayController is a class that inherits from ApplicationController, so it automatically gets all the default controller behavior. What does this code have to do? For now, it does nothing—we simply have empty action methods named hello() and goodbye(). To understand why these methods are named this way, we need to look at the way Rails handles requests.

Rails and Request URLs

Like any other web application, a Rails application appears to its users to be associated with a URL. When you point your browser at that URL, you are talking to the application code, which generates a response to you.

```
create    app/models/product.rb
invoke    test_unit
create      test/models/product_test.rb
create      test/fixtures/products.yml
invoke  resource_route
 route    resources :products
invoke  jbuilder_scaffold_controller
create    app/controllers/products_controller.rb
invoke    erb
create      app/views/products
create      app/views/products/index.html.erb
create      app/views/products/edit.html.erb
create      app/views/products/show.html.erb
create      app/views/products/new.html.erb
create      app/views/products/_form.html.erb
invoke    test_unit
create      test/controllers/products_controller_test.rb
invoke    helper
create      app/helpers/products_helper.rb
invoke      test_unit
create        test/helpers/products_helper_test.rb
invoke    jbuilder
 exist      app/views/products
create      app/views/products/index.json.jbuilder
create      app/views/products/show.json.jbuilder
invoke  assets
invoke    coffee
create      app/assets/javascripts/products.js.coffee
invoke    scss
create      app/assets/stylesheets/products.css.scss
invoke  scss
create    app/assets/stylesheets/scaffolds.css.scss
```

The generator creates a bunch of files. The one we're interested in first is the *migration* one, namely, 20121130000001_create_products.rb.

A migration represents a change we want to make to the data, expressed in a source file in database-independent terms. These changes can update both the database schema and the data in the database tables. We apply these migrations to update our database, and we can unapply them to roll our database back. We have a whole section on migrations starting in Chapter 22, *Migrations*, on page 367. For now, we'll just use them without too much more comment.

The migration has a UTC-based timestamp prefix (20121130000001), a name (create_products), and a file extension (.rb, because it's Ruby code).

The timestamp prefix you will see will be different. In fact, the timestamps used in this book are clearly fictitious. Typically your timestamps will not be consecutive; instead, they will reflect the time the migration was created.

```
demo> rails server
=> Booting WEBrick
=> Rails 4.0.0 application starting in development on http://0.0.0.0:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
[2013-04-18 20:22:16] INFO  WEBrick 1.3.1
[2013-04-18 20:22:16] INFO  ruby 2.0.0 (2013-02-24) [x86_64-linux]
[2013-04-18 20:22:16] INFO  WEBrick::HTTPServer#start: pid=25170 port=3000
```

Which web server is run depends on what servers you have installed. WEBrick is a pure-Ruby web server that is distributed with Ruby itself and therefore is guaranteed to be available. However, if another web server is installed on your system (and Rails can find it), the `rails server` command may use it in preference to WEBrick. You can force Rails to use WEBrick by providing an option to the `rails` command.

```
demo> rails server webrick
```

As the last line of the startup tracing indicates, we just started a web server on port 3000. The `0.0.0.0` part of the address means that WEBrick will accept connections on all interfaces. On Dave's OS X system, that means both local interfaces (127.0.0.1 and ::1) and his LAN connection. We can access the application by pointing a browser at the URL http://localhost:3000. The result is shown in Figure 1, *Newly created Rails application*, on page 18.

If you look at the window where you started the server, you'll see tracing showing you started the application. We're going to leave the server running in this console window. Later, as we write application code and run it via our browser, we'll be able to use this console window to trace the incoming requests. When the time comes to shut down your application, you can press Ctrl-C in this window to stop WEBrick. (Don't do that yet—we'll be using this particular application in a minute.)

At this point, we have a new application running, but it has none of our code in it. Let's rectify this situation.

2.2 Hello, Rails!

We can't help it—we just have to write a "Hello, World!" program to try a new system. Let's start by creating a simple application that sends our cheery greeting to a browser. After we get that working, we will embellish it with the current time and links.

As we'll explore further in Chapter 3, *The Architecture of Rails Applications*, on page 29, Rails is a Model-View-Controller framework. Rails accepts incoming requests from a browser, decodes the request to find a controller,

```
create  Rakefile
create  config.ru
  :      :      :
create  vendor/assets/stylesheets
create  vendor/assets/stylesheets/.keep
   run  bundle install
Fetching gem metadata from https://rubygems.org/..........
  :      :      :
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
work>
```

The command has created a directory named demo. Pop down into that directory, and list its contents (using ls on a Unix box or using dir under Windows). You should see a bunch of files and subdirectories.

```
work> cd demo
demo> ls -p
app/  config/    db/      Gemfile.lock  log/     Rakefile     test/  vendor/
bin/  config.ru  Gemfile  lib/          public/  README.rdoc  tmp/
```

All these directories (and the files they contain) can be intimidating to start with, but we can ignore most of them for now. In this chapter, we'll use only one of them directly: the app directory, where we'll write our application.

Examine your installation using the following command:

```
demo> rake about
```

If you get a Rails version other than 4.0.0, please reread Section 1.4, *Choosing a Rails Version*, on page 8.

This command will also detect common installation errors. For example, if it can't find a JavaScript runtime, it will provide you with a link to available runtimes.

If you see a bunch of messages concerning already initialized constants or a possible conflict with an extension, consider deleting the demo directory, creating a separate RVM gemset,[1] and starting over. If that doesn't work, use bundle exec[2] to run rake commands.

Once you get rake about working, you have everything you need to start a standalone web server that can run our newly created Rails application. So, without further ado, let's start our demo application.

1. https://rvm.io/gemsets/basics/
2. http://gembundler.com/v1.3/bundle_exec.html