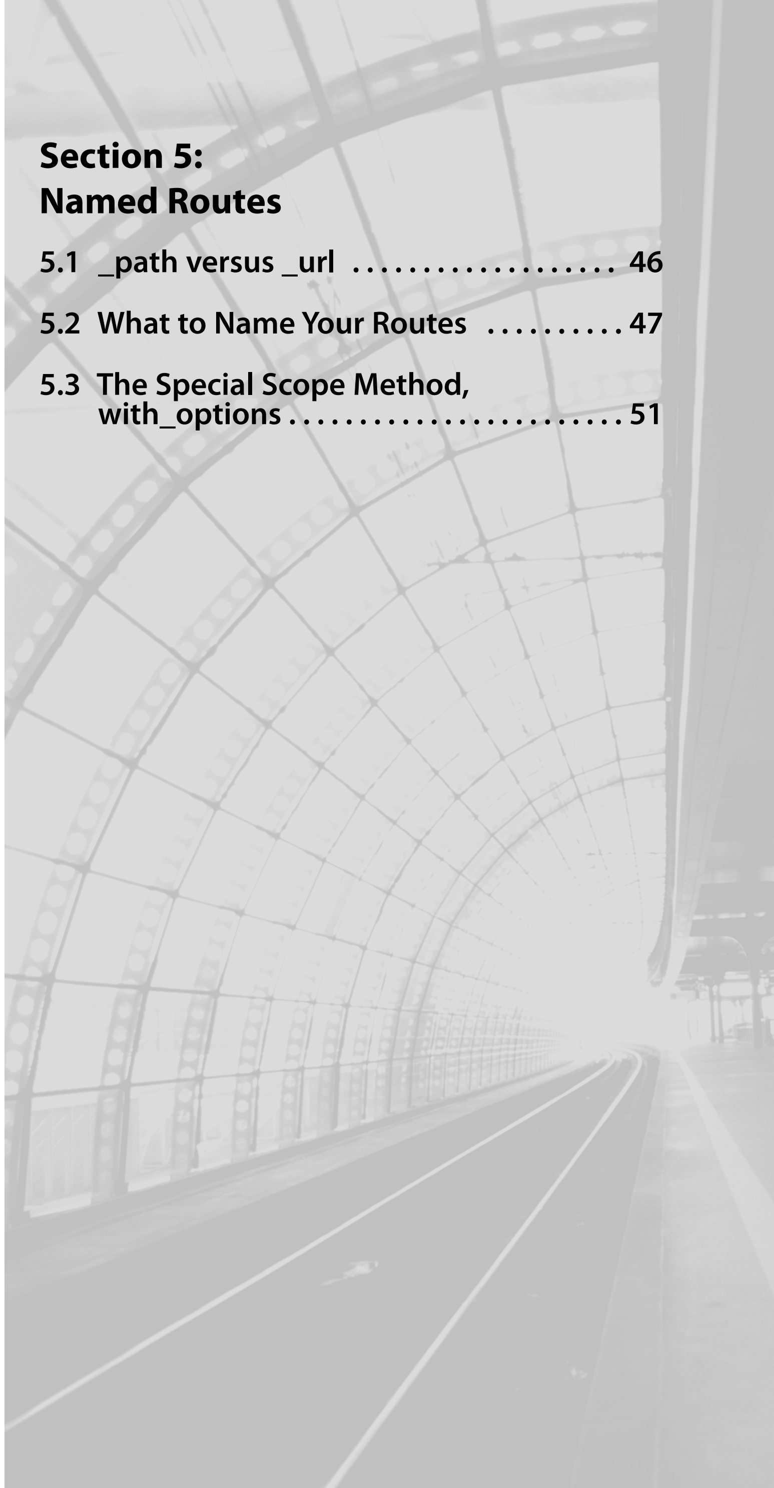

Section 5:
Named Routes

SECTION 5
Named Routes

Although this is a short section, the topic of named routes definitely deserves a section of its own. And what you learn here will feed directly into our examination of REST-related routing a little later.

The idea of naming a route is basically to make life easier on you, the programmer. When you name a route, a new method gets defined for use in your controllers and views; the method is called `*name*_url` (with *`name`* being the name you gave the route), and calling the method, with appropriate arguments, results in a URL being generated for the route. In addition, a method called `*name*_path` also gets created; this method generates just the path part of the URL, without the protocol and host components.

You name a route by calling a method on your mapper object with the name you want to use, instead of the usual connect:

```
map.help 'help',
  :controller => "main",
  :action => "show_help"
```

In this example, you'll get methods called `help_url` and `help_path` which you can use wherever Rails expects a URL or URL components:

```
<%= link_to "Help!", help_path %>
```

And of course, the usual recognition and generation rules are in effect. The pattern string consists of just the static string component `"help"`. The path you'll see in the hyperlink will therefore be:

```
/help
```

When someone clicks on the link, the `show_help` action of the main controller will be invoked.

Named routes save you some effort when you need a URL generated. A named route zeros in on the route you need, bypassing the matching process. That means you don't have to provide as much detail as you otherwise would. You have to provide a value for any wildcard parameter in the route's pattern string. But you don't have to go down the laundry list of hard-coded, bound parameters. The only reason for doing that when you're trying to generate a URL is to steer the routing system to the correct route. But when you use a named route, the system already knows which rule you want to apply.

5.1 _path versus _url

When you create a named route, you're actually creating at least two named routes. In the preceding example, the two routes are `help_url` and `help_path`.

The difference, as you might guess, is that the `_url` method generates an entire URL, including protocol and domain, whereas the `_path` method generates just the path part.

You need to use the _url version in your controllers, when you use a named route as an argument to redirect_to. (You can't redirect to a path.) In your views you can use either, but it's customary to use _path in most cases because it produces a shorter string and the user agent (browser or otherwise) should be able to infer the URL either from the headers, from a base element, or from the URL of the request which resulted in its receiving the current document.

As you read this short cut, and as you examine other code and other examples, the main thing to remember is that help_url and help_path are basically doing the same thing. I tend to use the _url style in general discussions about named route techniques, but to use _path in examples that occur inside view templates (e.g., with link_to and form_for). It's mostly a writing-style thing, based on the theory that the URL version is more general and the path version more specialized. In any case, it's good to get used to seeing both and getting your brain to view them as very closely connected.

5.2 What to Name Your Routes

The best way to figure out what named routes you'll need is to think from the top down; that is, think about what you want to write in your application code, and then create the routes that will make it possible.

Take, for example, this call to link_to:

```
<%= link_to "Auction of #{auction.item.description}",
    :controller => "auctions",
    :action     => "show",
    :id         => auction.id %>
```

It would be nice to shorten it—to do something like this instead:

```
<%= link_to "Auction for #{auction.item.description}",
      auction_path(:id => auction.id) %>
```

using the named route `auction_path`. After all, the routing rule:

```
map.connect "auctions/:id",
  :controller => "auctions",
  :action => "show"
```

already specifies the controller and action. It seems a little heavy-handed to spell all that information out again, just so that the routing system can figure out which route we want.

This is a good candidate for a named route. Giving the route a name is a shortcut; it takes us straight to that route, without a long search and without having to provide a thick description of the route's hard-coded parameters.

The named route will be the same as the plain route, except we replace `connect` with the name we want to give the route:

```
map.auction "auctions/:id",
  :controller => "auctions",
  :action => "show"
```

In the view, we can now use the more compact version of `link_to`; and we'll get (for auction 3, say) this URL in the hyperlink:

```
http://localhost:3000/auctions/show/3
```

5.2.1 A Bit of Argument Sugar

In fact, we can make the argument to `auction_path` even shorter. If you need to supply an `id` number as an argument to a named route, you can just supply the number, without spelling out the `:id` key:

```
<%= link_to "Auction for #{auction.item.description}",
      auction_path(auction.id) %>
```

And the syntactic sugar goes even further: You can just provide objects and Rails will grab the `id`:

```
<%= link_to "Auction for #{auction.item.description}", auction_path(auction) %>
```

This principle extends to other wildcards in the pattern string of the named route. For example, if you've got a route like this:

```
map.item 'auction/:auction_id/item/:id',
  :controller => "items",
  :action => "show"
```

you'd be able to call it like this:

```
<%= link to item.description, item_path(@auction, item) %>
```

and you'd get something like (depending on the exact `id` numbers):

```
/auction/5/item/11
```

as your path. Here, we're letting Rails infer the `id`s of both an `auction` object and an `item` object. As long as you provide the arguments in the order that their `id`s occur in the route's pattern string, the correct values will be dropped into place in the generated path.

5.2.2 A Little Sugar with Your Sugar?

Furthermore, it doesn't have to be the `id` value that the route generator inserts into the URL. You can override that value by defining a `to_params` method in your model.

Let's say you want the description of an item to appear in the URL for the auction on that item. In the item.rb model file, you would override `to_params`; here, we'll override it so that it provides a munged (stripped of punctuation and joined with hyphens) version of the description:

```
def munge_description
  description.gsub(/\s/, "-").gsub([^\W-], '').downcase
end

def to_param
  munge_description
end
```

Subsequently, this method call:

```
item_path(@item)
```

will produce something like this:

```
/auction/3/item/cello-bow
```

Of course, if you're putting things like "cello-bow" in a path field called `:id`, you will need to make provisions to dig the object out again. Blog applications that use this technique to create "slugs" for use in permanent links often have a separate database column to store the munged version of the title that serves as part of the path. That way, it's possible to do something like this:

```
Item.find_by_munged_description(params[:id])
```

to unearth the right item. (And yes, you can call it something other than `:id` in the route to make it clearer!)

5.3 The Special Scope Method, with_options

Sometimes you might want to create a bundle of named routes, all of which pertain to the same controller. You can achieve this kind of batch creation of named routes via the `with_options` mapping method.

Let's say you've got the following named routes:

```
map.help '/help', :controller => "main", :action => "help"
map.contact '/contact', :controller => "main", :action => "contact"
map.about '/about', :controller => "main", :action => "about"
```

You can consolidate these three named routes like this:

```
map.with_options :controller => "main" do ¦main¦
  main.help '/help', :action => "help"
  main.contact '/contact', :action => "contact"
  main.about '/about', :action => "about"
end
```

The three inner calls create named routes that are scoped—constrained—to use `"main"` as the value for the `:controller` parameter, so you don't have to write it three times.

Note that those inner calls use `main`, not `map`, as their receiver. Once the scope is set, `map` calls upon the nested mapper object, `main`, to do the heavy lifting.

At this point, we're on the cusp of REST.