



Addison-Wesley Professional Ruby Series



# THE MERB WAY

*Foreword by* **Obie Fernandez**, *Series Editor*

FOY SAVAS

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales  
international@pearson.com

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

*Library of Congress Cataloging-in-Publication Data*  
Savas, Foy.

The Merb way / Foy Savas.

p. cm.

ISBN 978-0-321-60638-9 (pbk. : alk. paper)

1. Web site development. 2. Merb (Electronic resource) I. Title.

TK5105.888.S2775 2009

006.7'6—dc22

2009013263

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.  
Rights and Contracts Department  
501 Boylston Street, Suite 900  
Boston, MA 02116  
Fax: (617) 671-3447

The code in this book may be distributed only subject to the terms and conditions set forth in the MIT License.

The MIT License reads:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation to the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

ISBN-13: 978-0-321-60638-9

ISBN-10: 0-321-60638-8

Text printed in the United States on recycled paper at Donnelley in Crawfordsville, Indiana.

First printing, June 2009

## **Editor-in-Chief**

Mark Taub

## **Acquisitions Editor**

Debra Williams Cauley

## **Development Editor**

Michael Thurston

## **Managing Editor**

John Fuller

## **Project Editor**

Anna Popick

## **Copy Editor**

Barbara Wood

## **Indexer**

Richard Evans

## **Proofreader**

Barbara Wood

## **Technical Reviewers**

Matthew Knox

Jen Lindner

## **Editorial Assistant**

Kim Boedigheimer

## **Cover Designer**

Chuti Prasertsith

## **Compositor**

ITC

# Foreword

---

I viewed the Merb project with a big dollop of suspicion almost from the very start. First of all, it started out as a really simple concept, but got a lot more complicated as additional contributors got involved. And whether my suspicion was misplaced or not, I couldn't shake the feeling that the Merb people were insisting on reinventing Ruby on Rails simply to be difficult.

Merb isn't *that* different from Rails, nor is it a major improvement as far as I was concerned. It was only *slightly* better (in some regards), and only *slightly* different from Ruby on Rails. Sure there were some benchmarks showing big performance gains for using Merb, but I was not convinced enough to switch. And I couldn't bear to use Merb and Rails concurrently. You see, they were very similar, yet different in subtle ways. The last thing I want to do when I'm programming on hard deadlines is to slow down to remember the differences between Merb's `render` method and the one in Rails, etc.

Life continued, and while I was very happy and making a ton of money working with and writing about Rails, I still occasionally looked at Merb, if only to see what they were up to. Sure enough, by 2008 my perception of Merb had started to change. I met Foy Savas and he gave me the gist of why it would make sense to write *The Merb Way*. I concurred and Foy got busy writing.

As Merb approached a 1.0 release, it appeared that Yehuda Katz and his band of merry discontents were actually achieving a viable platform, one that would attain permanent status as an alternative to Rails. I started hearing about significant production projects running on Merb, such as `yellowpages.com`. My appreciation for Merb grew, as I got familiar with the project goals and noticed how the competition was acting as a healthy stimulus for Rails to not rest on its laurels. I wasn't ready to switch to Merb myself, since my key criterion is maturity, which Rails has in spades, but no longer was I suspicious.

In November of 2008 I was extremely happy about Foy being almost finished writing the first manuscript of this book, knowing that it would be a good complement to the Professional Ruby Series and a good companion for *The Rails Way*. Foy delivered an awesome conference presentation in Boston where he covered some of the key concepts from the book in an engaging and entertaining manner—the crowd ate it up and I knew *The Merb Way* would be a winner.

It wasn't too many weeks later that the unthinkable happened. After an eruption of feuding between members of the Rails and Merb teams, some secret meetings occurred and shocking news was unveiled right before Christmas: DHH and Yehuda happily proclaimed that the core teams of both frameworks were merging and that within a year, the Merb codebase would be merged with Rails in order to produce Rails Version 3, after which development of Merb would discontinue.

The million-dollar question (okay, not quite that much) for Foy and myself as the Series Editor was whether to continue with the publication of *The Merb Way*. After all, if the Merb framework was going away, then what was the point?

After letting the matter settle for a while, we arrived at our answer, which you already know since you are holding this book. It turns out that learning about Merb is valuable in a number of important ways. First, Merb is still used rather widely and we suspect that its lifetime will exceed what the current core team has in mind. Second, Foy is a gifted writer and his descriptions of the philosophies that impacted the design and implementation of Merb are definitely useful to everyone who will be affected by the changes that will happen in Rails 3.

It's with great pleasure that I welcome the incomparable Foy Savas and publish *The Merb Way* as a full-fledged and proud installment of the Professional Ruby Series. I sincerely hope you get as much out of this book as I have.

Obie Fernandez  
March 31, 2009

# Introduction

---

道可道非常道  
名可名非常名  
—老子

A way that can be taken rarely stays the way.  
A name that can be given rarely stays the name.  
—Laozi  
(translated by your author)

The first two lines of the *Dao De Jing* capture truths so fundamental that we find them everywhere—including, without exception, in this book. Originally entitled the *The Merb Companion*, it was intended to be the advanced practitioner’s bible. But the little framework named Merb grew quickly, so much so that the Rails world took notice, and before long, the core teams came together and decided to merge the two. Yet despite Merb’s fundamentals being seen as the basis for the features of the future, Rails conventions needed to be maintained long-term. Incidentally, this book’s content was left in an awkward limbo. Should we wait for the completion of the merge or perhaps publish immediately? Should we preserve the chapters we had written so far or produce them anew after the merge? Soon, however, an epiphany came. Though the book had material highly relevant to the Merb developers of the day and was equally suitable for bringing foresight to the Rails developers of tomorrow, its greatest and lasting potential was in shedding light on how Merb, the framework that even Rails envied, had been designed. Thus, we arrive here, at the introduction of *The Merb Way*.

## Born a pastie

Zoom back to September 9, 2006. If you were using Rails, it was version 1.1. As for Git, it was still just something the Linux kernel used. So naturally, when Ezra Zygmuntowicz needed to publicly distribute the then only 120-line source code for Merb, a pastie made the most sense. Turn back a few pages and you'll find that pastie, prepended to this book in all of its glory. Look it over once and you'll realize Ezra had an itch to scratch: fast Ruby template rendering with the smallest possible memory footprint. To do this he used only two gems, Mongrel and ERB, which, for those keen on etymology, also serve as the roots of the name Merb itself. But like any project that starts off both small and practical, it grew. A few months later a gem was put up on RubyForge. For a while it came with an example app that touted Merb's ability to handle file uploads while not locking out other requests. This became the most popular use of Merb, often appearing coupled with Rails applications in the wild.

## Created by rebels

But Merb has moved far beyond the stage when its application generator could produce the `uploader` application. This may have been because it quickly became a breeding ground in which Rails concepts could be rethought. Eventually, this led to a shared vision among developers that Merb would be Rails done right. Thankfully, the openly opinionated nature of Rails actually kept them away from simply adopting alternative opinions. Instead, they chose as one of their maxims agnosticism, in the form of opt-in modularity. This allowed Merb application developers to work with whatever tools they needed for their projects. Merb's versatility meant that it would become the underlying framework for numerous customized stacks that otherwise did not easily fit within the opinions of Rails. Its developers, consequently, have often been thought of as rebels from the Rails mentality, since many of them maintain strong opposing views on particular Rails opinions.

However, that's not all there is to the Merb development story. The crowd of developers also had an arguably more scientific bent, aiming to make the best selection of the methods they employed through microbenchmarks. Ruby is often cited as one of the world's slowest programming languages from a computational perspective, but to the developers of Merb, this didn't mean that microseconds didn't matter. After all, web development frameworks are often tested in terms of responses per second. So when it came to Merb internals, whether it was in route matching or filter chaining, only the fastest Ruby constructs were used. The end result was Merb blazing past Rails

in response time benchmarks. More profoundly, Merb raised the bar for Ruby web framework performance in general, finally allowing Ruby to win when pitted against other frameworks from different languages.

Interestingly enough, this combination of performance and modularity is also indicative of another Merb development character trait: its desire to solve higher-level problems. Whereas Rails features have nearly consistently been born of necessity, Merb's focus has been on abstract goals like versatility and design. This has pushed the limits of what can be done with Merb, resulting in advancements like the abstract controller class, code slices, and arbitrary application layout. In sum, Merb in many ways conceptually broke the "frame" in framework by thinking of itself less as a way to produce Model-View-Controller (MVC) web apps and more as a highly modular platform upon which nearly all Ruby web development, no matter how unusual, can occur.

## The future of Rails

Put all this goodness together, and you're bound to attract the attention of most developers. Not surprisingly, teams large and small began using Merb as a full application framework in production environments for their most demanding applications. This finally put Merb to the test, and the results were completely in line with its benchmarks: more responses per second with a solidly smaller memory footprint. All of this occurred at the cusp of its reaching its 1.0 release on November 7, 2008.

Quickly the Merb development team oriented itself toward larger goals through what would have been a 2.0 release. Objectives included, among other things, an abstracted Object Relational Mapper (ORM) that could be used to create an agnostic admin panel and convenient direct routing to templates. But with the stable release already out, developers at large began taking Merb much more seriously, and soon online comment wars fanned flames between the Rails and Merb core teams. Thus prompted to confront each other, the core development teams began to talk it over. However, soon they began earnestly asking themselves whether the large amount of duplication between the two frameworks was warranted. At the end they concluded two things: first, that each side had aspects that would be beneficial to the other, and second, that yes, they could work together. Announced officially on December 23, 2008, the Rails and Merb development effort would be merged, with a promise of a smooth transition from both Merb 1 and Rails 2 to the combined Rails 3.

So what can we expect from this merger? Well, above all will be the fusion of attitudes and objectives from both sides. The Rails of the future will undoubtedly appeal to developers in both pragmatic and idealistic ways. Merb will bring speed, performance,

and modularity along with a tempering of the preestablished Rails opinions. We will see a flowering of custom Rails stacks, some of which may eventually push the limits of the Rails framework itself. Toward what—who knows? But with the possibility of such a greatly expanded domain, two things are certain. The first is that anyone in software who needs to build an agile and sophisticated web application upon a high-performance framework should definitely get on this train. The second, which applies more to developers themselves, is that a firm understanding of the design decisions behind Merb's development will not only prepare them for the future of Rails but can also open up their own talents through the elegance of its code.

This book is thus focused on giving developers the deepest possible understanding of Merb itself. While you may use it as a reference guide in the development of Merb applications (which we strongly recommend without hesitance until the release of Rails 3), you may also appreciate its guided exploration of the Merb source. That said, for those afraid of code, it may perhaps be a good time to put this book back on the shelf. Everyone else, be warned; we're going to cut deep into the framework itself, revealing not only what was done, but what can be learned from it. As you move forward, do not lose focus. After all, they say that one should always aim to learn from the best, and through Merb, the framework that without exaggeration brought the Rails monoculture to its knees, you are doing just that.



## CHAPTER 5

---

# Models

Models are related data and algorithms that respectively represent the properties and behaviors of domain objects used within an application. Models are used in nearly every decently complex object-oriented application because they organize application data in object form. Within the scope of a typical Merb application, examples of modeled objects may include users, posts, entries, or comments. These objects are most often persisted in a database through the use of an Object Relational Mapper, or ORM. As agnosticism has been central to Merb's design, application developers are free to integrate whatever Ruby ORM they may need. Nonetheless, the default ORM included as part of the standard Merb stack is DataMapper, an ActiveRecord-like ORM that aims to push the boundaries of object-data interaction even more significantly toward the object side.

Given its acceptance as part of the standard Merb stack, we will cover the use of only DataMapper in this chapter. However, don't let that stop you from using ActiveRecord, Sequel, or any of the other options, since each of these ORMs is capable of creating, retrieving, updating, and deleting persisted data and is fully supported by the Merb core.

### 5.1 Configuration

Prior to being able to use the DataMapper ORM within your Merb application, you will have to make sure that you have done three things:

- Included the DataMapper dependencies within `config/dependencies.rb`
- Selected DataMapper as the ORM in `init.rb`
- Configured your database connection in `config/database.yml`

However, if you used `merb-gen` to generate a standard application and have SQLite3 installed, then you'll find that all of these are already covered for you. If your application was generated in some other manner, you may want to see Chapter 1 to figure out how your configuration files should be laid out.

### Other standard dependencies

The standard Merb generator inserts several other DataMapper dependencies into the file `config/dependencies.rb`. These commonly used DataMapper plugins provide functionality such as migrations, automatic timestamping, validations, and record aggregation. You are free to comment out or remove any of these dependencies, but as we'll see through examples, even the most basic application models tend to want them to be available.

DataMapper works with several different database back ends. Support for MySQL, PostgreSQL, and SQLite3 are all included via the DataMapper core. If you're using another database application, you may be able to find an appropriate adapter in the `dm-more` or `dm-adapters` gems. Alternatively, if your computer has the memory for it, there's a DataMapper adapter that will store your model records completely in memory.

### Not just databases

When you use DataMapper, you aren't limited to interacting only with conventional databases. Instead, the DataMapper concept of a repository applies equally well to just about any data source composed of records. This includes numerous web service APIs where CRUD-like operations are available. A great example of such an adapter is the Salesforce adapter Yehuda Katz has written. In just under 300 lines of code, the adapter provides full DataMapper interaction via the Salesforce API. You'll find that these adapters can also be used to augment models through secondary repositories, which is commonly the case when using Bernerd Schaefer's full-text-search Ferret adapter.

Repository configuration is done in a YAML file named `config/database.yml`. The standardly generated `config/database.yml` file is broken into four environments—development, test, production, and rake—corresponding to the environments that are standardly used by a Merb application. If you have another environment—for instance, a staging environment—it, too, can be listed in this file.

Typically there is overlap in the configuration of environments, so YAML node anchors and aliases are used to reduce redundancy.

### What's a YAML node?

YAML nodes are named blocks of content in a YAML file. You can set up an anchor to the content of a node by using an ampersand and word after the naming of the node. Below we connect the content of `development` with the anchor `&default`.

```
development: &default
  adapter: mysql
  database: example_development
```

We can reference this anchor by using the alias `*default` later on. If we use this alias as the value on a merge key, `<<`, it will include the previously defined content. Here we merge in the defaults from before and then override the value for the database:

```
test:
  <<: *default
  database: example_test
```

## 5.2 Model classes

Model classes exist in the directory `app/models`. These classes typically have singular names and exist one class per file. Examples that may appear in a Merb blog application are `Post` in the file `app/models/post.rb` and `FavoriteLink` in `app/models/favorite_link.rb`. Organizing model classes in this way isn't strictly necessary, though, since Merb by default includes everything inside `app/models` by recursive glob. In other words, so long as you make sure that your model class is somewhere in `app/models`, it will be available at boot. Nonetheless, our recommendation is that you do arrange your model classes in a reasonable way, so that both you and others can easily find them.

While various `DataMapper` modules can incorporate different functionality within your model class, the fundamental module that must be included in order for your class to work with `DataMapper` is `DataMapper::Resource`. Below we include this module and effectively set up our `User` class as a `DataMapper` model.

```
class User
  include DataMapper::Resource
end
```

### Design decision: resource module versus base class

DataMapper model classes are not created through inheritance from an abstract base class. If you've used ActiveRecord, which does do so, you may wonder why this alternate design decision has been made. Interestingly enough, early versions of DataMapper did actually use a parent class called `Mapper::Base`. With time, however, the developers of DataMapper grew concerned that this was conceptually coupling application logic with DataMapper's own. In other words, application models are more reasonably thought of as being enhanced through database persistence rather than as stemming off a library's base class that provides such functionality.

In some ways, this design decision can be seen as an application of the Principle of Substitutability, which insists that subtypes are capable of replacing their parents without affecting the correctness of the program. While this principle doesn't exactly translate well when using Ruby (proving correctness through types isn't anywhere near Ruby's slant), we can learn from the principle by recognizing that inheritance is best used when a subclass makes no behavioral constraining modifications on the parent.

Let's take a look at the source behind the `Resource` module to get a feel for how it affects the class in which it is included:

```
module DataMapper
  module Resource

    # ...

    # @api public
    def self.append_inclusions(*inclusions)
      extra_inclusions.concat inclusions
      true
    end

    def self.extra_inclusions
      @extra_inclusions ||= []
    end

    # When Resource is included in a class this
    # method makes sure it gets all the methods
    #
    # -
    # @api private
    def self.included(model)
      model.extend Model
      model.extend ClassMethods
    end
```

```

    if defined?(ClassMethods)
      model.const_set('Resource', self)
      unless model.const_defined?('Resource')
        extra_inclusions.each { |inclusion|
          model.send(:include, inclusion) }
      descendants << model
      class << model
        @_valid_model = false
        attr_reader :_valid_model
      end
    end

    # ...
  end
end

```

Focusing on the last method listed above, we can see that the `Resource` module extends the class in which it is included (above called `model`). In the process, it principally extends it with the `Model` module, which contains the logic for property persistence and object retrieval. As we come to these methods later on, we'll open up the `Model` class source as well. For now, however, take note of `Resource`'s ability to include extra modules through the class method `append_inclusions`. This method is used extensively by `DataMapper` plugins that need to extend the functionality of all `DataMapper` models. For instance, below is some of the source for `dm-timestamps`, which automatically sets timestamps on properties of particular names.

```

module DataMapper
  module Timestamp
    Resource.append_inclusions self

    # ...

    def self.included(model)
      model.before :create, :set_timestamps
      model.before :update, :set_timestamps
      model.extend ClassMethods
    end

  end
end

```

The first line appends the module to all `DataMapper` models. Later on, using its own `self.included`, the module includes its own logic into the model. This cascading of modules is thus particularly effective for code as modular as `DataMapper`.

## 5.3 Properties

Each DataMapper model is able to persist its data. The kind of data it is able to store is defined through its properties. If you're using a typical database, these properties correlate with the columns of the model's corresponding table. Below is an example of a DataMapper model with three properties.

```
class TastyAnimal
  include DataMapper::Resource

  property :id, Serial
  property :name, String
  property :endangered, TrueClass

end
```

In many ways, you can think of properties as persistent accessors. In fact, taking a look into the source of the property method (found in the `Model` resource we spoke about earlier), we find that a dynamic getter and setter are created using `class_eval`:

```
def property(name, type, options = {})
  property = Property.new(self, name, type, options)

  create_property_getter(property)
  create_property_setter(property)

  # ...
end

# ...

# defines the getter for the property
def create_property_getter(property)
  class_eval <<-EOS, __FILE__, __LINE__
    #{property.reader_visibility}
    def #{property.getter}
      attribute_get(#{property.name.inspect})
    end
  EOS

  # ...

end

# defines the setter for the property
def create_property_setter(property)
  unless instance_methods.include?("#{property.name}")
```

```

class_eval <<-EOS, _ _FILE_ _, _ _LINE_ _
  #{property.writer_visibility}
  def #{property.name}=(value)
    attribute_set("#{property.name.inspect}", value)
  end
EOS
end
end

```

The most important thing to learn from the source shown above is that properties dynamically create getter and setter methods. Additionally, these methods can end up protected or private through visibility attributes. Finally, the getters and setters produced are not exactly equivalent to `attr_reader` and `attr_writer` because of their internal use of the methods `attribute_get` and `attribute_set`.

Going back to the `Resource` source, we can find these two methods manipulating the values of model properties, once again located in `Model`. You'll have to excuse this volleying back and forth, but the point of the `Resource` and `Model` modules is to separate individual resource methods from those related to the model as a whole.

```

# @api semiplugin
def attribute_get(name)
  properties[name].get(self)
end

# @api semipublic
def attribute_set(name, value)
  properties[name].set(self, value)
end

protected

def properties
  model.properties(repository.name)
end

```

You may have noticed the `@api semipublic` comment above the getter and setter methods. This is because application developers should not ordinarily need to use these methods. Plugin developers, on the other hand, may need to use them as the easiest way to get and set properties while making sure they are persisted.

For application developers, however, this does bring up one important point: Do not use instance variables to set property values. The reason is that while this will set the object's value, it will unfortunately short-circuit the model code that is used to track whether a property is dirty. In other words, the property value may not persist later upon

save. Instead, you should use the actual property method. Below you'll find an example with comments that should get the point across.

```
class Fruit
  include DataMapper::Resource

  property :id, Serial
  property :name, String
  property :eaten, TrueClass

  def eat
    unless eaten?
      # will not persist upon save
      @eaten = true

      # will persist upon save
      eaten = true
    end
  end
end
```

Before we describe the extended use of properties, let's take a look at the database side to understand how persistence works.

### 5.3.1 Database storage

In order to persist the data of model objects, we need to set up our database for that data to be stored. The default-generated configuration files use a SQLite3 database file called `sample_development.db`. This setup is perfect for most development scenarios given its quickness to get up and running. With that in mind, we'd say stick with it whenever possible, leaving the alteration of `config/database.yml` for production or staging environments.

#### Create your database with rake

If you are using MySQL or PostgreSQL for development, though, it may be useful to know that there is a Merb DataMapper rake task that can create your database for you. After making sure your `config/database.yml` file contains the appropriate username and password, `rake db:create` issues the database admin command you may have otherwise forgotten.



### 5.3.1.1 Automigrating the DB schema

Databases typically need to be prepped for the data they will store during application development. The process by which DataMapper does this is called **automigration**, because DataMapper uses the properties listed in your models to automatically create your database schema for you. Using the provided Merb DataMapper rake task, we can automigrate the model that we created earlier and then take a peek inside the database to see what was done:

```
$ rake db:automigrate
$ sqlite3 sample_development.db
sqlite> .tables
tasty_animals
sqlite> .schema
CREATE TABLE "tasty_animals" ("id" INTEGER NOT NULL
  PRIMARY KEY AUTOINCREMENT, "name" VARCHAR(50), "
  is_endangered" BOOLEAN);
```

As you can see, a table with a pluralized and snake-cased name was created for our model, `TastyAnimal`. Remembering the various properties of the model class, we can also spot corresponding columns inside the schema's `CREATE` statement. Note that while Ruby classes were used on the property lines, standard SQL types appear in the database.

#### Design decision: emphasize the model as an object

DataMapper has notably chosen to include properties within the model files. Those familiar with ActiveRecord may find this odd, having been accustomed to distributing schema information across migration files. There are, however, a number of benefits to the DataMapper way of handling properties, including

- An explicitness that helps developers understand how a model is used
- An ability to work around legacy systems far more easily due to the lack of schema dependence
- The possibility of distributing model property persistence across multiple databases or repositories
- The time-saving nature of automigrations

These benefits may take a while to appreciate given their subtleness, but there is a more general paradigm shift going on that we should mention: The inclusion and use of properties in DataMapper model classes essentially pushes the ORM concept even further object-side, where the resulting combination of less magic and less maintenance is a total win.

The code behind automigration is definitely worth studying, so let's take a look at the module `AutoMigrations`, which includes itself within the `Model` module:

```

module DataMapper
  module AutoMigrations
    def auto_migrate!(repository_name =
      self.repository_name)

      auto_migrate_down!(repository_name)
      auto_migrate_up!(repository_name)
    end

    # @api private
    def auto_migrate_down!(repository_name =
      self.repository_name)

      # repository_name ||= default_repository_name
      repository(repository_name) do |r|
        r.adapter.destroy_model_storage(r,
          self.base_model)
      end
    end

    # @api private
    def auto_migrate_up!(repository_name =
      self.repository_name)

      repository(repository_name) do |r|
        r.adapter.create_model_storage(r,
          self.base_model)
      end
    end

    def auto_upgrade!(repository_name =
      self.repository_name)

      repository(repository_name) do |r|
        r.adapter.upgrade_model_storage(r, self)
      end
    end

    Model.send(:include, self)

  end # module AutoMigrations
end # module DataMapper

```

As you can see, there are two API public class methods you can use with models, `auto_migrate!` and `auto_upgrade!`. These effectively call the three adapter methods `destroy_model_storage`, `create_model_storage`, and `upgrade_model_storage`. Let's go deep into the source and see how these three methods do the heavy lifting:

```
class DataMapper::Adapters::AbstractAdapter
  module Migration

    def upgrade_model_storage(repository, model)
      table_name = model.storage_name(repository.name)

      if success = create_model_storage(repository,
        model)

        return model.properties(repository.name)
      end

      properties = []

      model.properties(repository.name).
        each do |property|

          schema_hash = property_schema_hash(repository,
            property)

          next if field_exists?(table_name,
            schema_hash[:name])

          statement = alter_table_add_column_statement(
            table_name, schema_hash)

          execute(statement)
          properties << property
        end

      properties
    end

    def create_model_storage(repository, model)
      return false if storage_exists?(
        model.storage_name(repository.name))

      execute(create_table_statement(repository, model))
    end
  end
end
```

```

    # ... create indexes

    true
end

def destroy_model_storage(repository, model)
  execute(drop_table_statement(repository, model))
  true
end

end
end

```

The simplest of these, `destroy_model_storage`, executes a drop table statement. The `create_model_storage` method, on the other hand, first checks to see if the model storage already exists, returning false if it does or true if it does not, and consequently has the chance to create it. Finally, `upgrade_model_storage` is the most complicated of the three. It first attempts to create the storage (effectively testing whether it exists or not) and then attempts to add new columns for new properties. This leaves existing data in place and is perfect if you have simply added properties to a column. Lest this appear to be no more than hand waving, let's dig even deeper into the methods that the `AbstractAdapter` uses to create the SQL for these statements:

```

class DataMapper::Adapters::AbstractAdapter

  # immediately following the previous code

  module SQL
    private

    def alter_table_add_column_statement(table_name,
      schema_hash)

      "ALTER TABLE "+
      quote_table_name(table_name) +
      "ADD COLUMN "+
      property_schema_statement(schema_hash)
    end

    def create_table_statement(repository, model)
      repository_name = repository.name

      statement = <<-EOS.compress_lines
        CREATE TABLE
          #{quote_table_name(

```

```

        model.storage_name(repository_name))}

    ({model.properties_with_subclasses(
        repository_name).map { |p|

        property_schema_statement(
            property_schema_hash(repository, p))

        } * ', ' })
EOS

    if (key = model.key(repository_name)).any?
        statement << ", PRIMARY KEY({ key.map { |p|
            quote_column_name(p.field(repository_name))
        } * ', ' })"
    end

    statement << ')'
    statement
end

def drop_table_statement(repository, model)
    "DROP TABLE IF EXISTS "+
    quote_table_name(model.storage_name(
        repository.name))
end

def property_schema_hash(repository, property)
    schema = self.class.type_map[property.type].
        merge(:name => property.field(repository.name))

    if property.primitive == String &&
        schema[:primitive] != 'TEXT'
        schema[:size] = property.length
    elsif property.primitive == BigDecimal ||
        property.primitive == Float
        schema[:precision] = property.precision
        schema[:scale] = property.scale
    end

    schema[:nullable?] = property.nullable?
    schema[:serial?] = property.serial?

    if property.default.nil? ||
        property.default.respond_to?(:call)
        schema.delete(:default)
        unless property.nullable?

```

```

    else
      if property.type.respond_to?(:dump)
        schema[:default] = property.type.dump(
          property.default, property)
      else
        schema[:default] = property.default
      end
    end
  end

  schema
end

def property_schema_statement(schema)
  statement = quote_column_name(schema[:name])
  statement << " #{schema[:primitive]}"

  if schema[:precision] && schema[:scale]
    statement << "(#{[ :precision, :scale ].map {
      |k| quote_column_value(schema[k])
    } * ', '})"
  elsif schema[:size]
    statement << "(" +
      quote_column_value(schema[:size]) + ")"
  end

  statement << ' NOT NULL'
  unless schema[:nullable?]
    statement << " DEFAULT " +
      quote_column_value(schema[:default]) if
        schema.has_key?(:default)
  end
end

include SQL
end

```

The first thing you may notice is that the methods are included within a module called `SQL` and that the module is included immediately after it is closed. The reason for this is that within `DataMapper` adapters, code is often organized by use, and thus the encapsulation of private methods into a module easily allows for alternating regions of public and then private methods.

Now, turning to the actual methods, we can see that some of them—for instance, `drop_table_statement`—are just a line of simple SQL. Likewise, `alter_table_column_statement` is just a single line that outputs add column statements.

The `create_table_statement`, however, is far more complex, relying on various other methods to get its work done. One of these, `properties_with_subclasses`, pulls up all model properties, including those that are simply keys used with relationships. We'll go further into `properties_with_subclasses` later on when we examine model relationships, but for now let's take a look at the method `property_schema_statement`, which quotes the property as a column name and then appends its type. It also adds the appropriate SQL for decimals, non-nullables, and default values.

We hope this has brought you deep enough into the inner workings of automigration to both appreciate its design and get a feel for how adapter code handles the production of SQL more generally. But it would also be nice to be able to use some of it practically, and thankfully you can do so. For instance, if you're in mid-development, you may fire up interactive Merb and use `auto_upgrade!` on a model to which you've added properties:

```
> Fruit.auto_upgrade!
```

Likewise, you may want to refresh the data of a model using `auto_migrate!` in the middle of a test file. Here's an example we've spotted in the wild:

```
before :each do
  Invite.auto_migrate!
end
```

## 5.3.2 Defining properties

Let's now take a more rigorous look at properties as well as the options we have while defining them. As we've seen, each property is defined on its own line by using the method `property`. This class method is mixed in via the inclusion of `Mapper::Resource`. It takes a minimum of two arguments, the first being a symbol that effectively names the property and the second being a class that defines what type of data is to be stored. As we will see soon, an optional hash of arguments may also be passed in.

### 5.3.2.1 Property types

While abstracting away the differences across database column types, `Mapper` has chosen to stay true as much as possible to using Ruby to describe properties types. Below is a list of the various classes supported by the `Mapper` core. Note that the inclusion of `Mapper::Resource` will include `DM` in your model class, and that when defining properties, you will not have to use the module prefix `DM::` before those that use it.

- `Class`—stores a Ruby `Class` name as a string. Intended for use with inheritance, primarily through the property type `DM::Discriminator`.
- `String`—stores a Ruby `String`. Default maximum length is 50 characters. Length can be defined by the optional hash key `:length`.
- `Integer`—stores a Ruby `Integer`. Length can be defined by the optional hash key `:length`.
- `BigDecimal`—stores a Ruby `BigDecimal`, intended for numbers where decimal exactitude is necessary. Can use the option hash keys `:precision` and `:scale`.
- `Float`—stores a Ruby `Float`. Primarily intended for numbers where decimal exactitude is not critical. Can use the two options hash keys `:precision` and `:scale`.
- `Date`—stores a Ruby `Date`.
- `DateTime`—stores a Ruby `DateTime`.
- `Time`—stores a Ruby `Time`.
- `Object`—allows for the marshaling of a full object into a record. It is serialized into text upon storage and when retrieved is available as the original object.
- `TrueClass`—a Boolean that works with any of the values in the array `[0, 1, 't', 'f', true, false]`. In MySQL it translates down to a `tinyint`, in PostgreSQL a `bool`, and in SQLite a `boolean`.
- `DM::Boolean`—an alias of `TrueClass`. This is around for legacy `DataMapper` support, simply to provide a more commonly recognized name for the type.
- `Discriminator`—stores the model class name as a string. Used for single-table inheritance.
- `DM::Serial`—used on the serial ID of a model. Serial IDs are auto-incremented integers that uniquely apply to single records. Alternatively, a property can use the `Integer` class and set `:serial` to `true`. You will nearly always see this type applied to the `id` property.
- `DM::Text`—stores larger textual data and is notably lazy-loaded by default.

You may be interested in knowing how the casting in and out of property values works. For the primitive types, values coming out of the database are cast using the method `Property#typecast`. Below we see how this method prunes results, modifying them into what we want in Ruby.



```

def typecast(value)
  return type.typecast(value, self) if type.respond_to?(:typecast)
  return value if value.kind_of?(primitive) || value.nil?
begin
  if primitive == TrueClass
    %w[ true 1 t ].include?(value.to_s.downcase)
  elsif primitive == String
    value.to_s
  elsif primitive == Float
    value.to_f
  elsif primitive == Integer
    value_to_i = value.to_i
    if value_to_i == 0
      value.to_s =~ /^(0x|0b)?0+/ ? 0 : nil
    else
      value_to_i
    end
  elsif primitive == BigDecimal
    BigDecimal(value.to_s)
  elsif primitive == DateTime
    typecast_to_datetime(value)
  elsif primitive == Date
    typecast_to_date(value)
  elsif primitive == Time
    typecast_to_time(value)
  elsif primitive == Class
    self.class.find_const(value)
  else
    value
  end
rescue
  value
end
end

```

Custom types, however, are handled by subclasses of an abstract type class called `DataMapper::Type`. These load and dump data in whatever way they are programmed to do. We'll see custom types later on when we examine some `DataMapper`-type plugins, but for now let's take a look at one of the custom types from the `DataMapper` core, `Serial`:

```

module DataMapper
  module Types
    class Serial < DataMapper::Type
      primitive Integer
    end
  end
end

```

```

    serial true
  end # class Text
end # module Types
end # module DataMapper

```

Note its use of the methods `primitive` and `serial`, which are defined in the class `DataMapper::Type`:

```

class DataMapper::Type
  PROPERTY_OPTIONS = [
    :accessor, :reader, :writer,
    :lazy, :default, :nullable, :key, :serial, :field,
    :size, :length, :format, :index, :unique_index,
    :check, :ordinal, :auto_validation, :validates,
    :unique, :track, :precision, :scale
  ]

  # ...

  class << self

    PROPERTY_OPTIONS.each do |property_option|
      self.class_eval <<-EOS, __FILE__, __LINE__
      def #{property_option}(arg = nil)
        return @#{property_option} if arg.nil?

        @#{property_option} = arg
      end
    EOS
  end

  def primitive(primitive = nil)
    return @primitive if primitive.nil?
    @primitive = primitive
  end

  # ...

end
end

```

From this we can first see that the `primitive` method sets the type to which the property value should be dumped. The `serial` method, on the other hand, is an example of the property option, which we're about to address.

### 5.3.2.2 Option hash

The third argument that the `property` method can take is an option hash, which affects various behavioral aspects of the property. For instance, below we've specified that a property should default to some value.

```
class Website
  include DataMapper::Resource

  property :id, Serial
  property :domain, String
  property :color_scheme, String, :default => 'blue'
end
```

Here's a list of the various property options and their uses:

- `:accessor`—takes the value `:private`, `:protected`, or `:public`. Sets the access privileges of the property as both a reader and a writer. Defaults to `:public`.
- `:reader`—takes the value `:private`, `:protected`, or `:public`. Sets the access privileges of the property as a reader. Defaults to `:public`.
- `:writer`—takes the value `:private`, `:protected`, or `:public`. Sets the access privileges of the property as a writer. Defaults to `:public`.
- `:lazy`—determines whether the property should be lazy-loaded or not. Lazy-loaded properties are not read from the repository unless they are used. Defaults to `false` on most properties, but is notably `true` on `DM::Text`.
- `:default`—sets the default value of the property. Can take any value appropriate for the type.
- `:nullable`—if set to `true` it will disallow a null value for the property. When `dm-validations` is used this invalidates a model.
- `:key`—defines a property as the table key. This allows for natural keys in place of a serial ID. This key can be used as the index on the model class in order to access the record.
- `:serial`—sets the property to be auto-incremented as well as to serve as the table key.
- `:field`—manually overrides the field name. Best used for legacy repositories.
- `:size`—sets the size of the property type.
- `:length`—alias of `:size`.

- `:format`—used with the `String` property type. When used with a `dm-validations` format can set a regular expression against which strings must validate.
- `:index`—sets the property to be indexed for faster retrieval. If set to a symbol instead of `true`, it can be used to create multicolumn indexes.
- `:unique_index`—defines a unique index for the property. When used with `dm-validations`, new records with nonunique property values are marked invalid. If set to a symbol instead of `true`, it can be used to create multicolumn indexes.
- `:auto_validation`—when used with `dm-validations`, can be used to turn off autovalidations by using the value `true`.
- `:track`—determines when a property should be tracked for dirtiness. Takes the values `:get`, `:set`, `:load`, and `:hash`.
- `:precision`—sets the number of decimal places allowed for `BigDecimal` and `Float` type properties.
- `:scale`—sets the number of decimal places after the decimal point for `BigDecimal` and `Float` type properties.

## 5.4 Associations

DataMapper also supports the defining of associations between models. If you've used ActiveRecord before, these are nearly the same. Otherwise, know that associations allow you to define the relationships between models (one-to-one, one-to-many, etc.), automatically creating database keys where necessary while also making it possible to conveniently pull up related model objects from associates. Let's survey the various relationships possible with DataMapper.

### Design decision: generalized association methods

The largest distinction between DataMapper and ActiveRecord associations is DataMapper's use of only two methods, `has` and `belongs_to`. This differs from ActiveRecord's one method per association type and has benefited DataMapper through simplicity of both internal design and interface. We'll see the elegance of the internal design later on, but know that these two generalizations allow application developers to simplify the cognitive steps needed to arrive at the appropriate associations. That is, to determine if an association should use `has` or `belongs_to`, you only need to ask whether a related model key should maintain a property for the ID of the other model. Moving from here, you can then define the cardinality and pathway of the

relationship if necessary. This may not seem like anything special to an experienced developer, but we've found that this easing of an interface's learning curve tends to be indicative of cleaner design, making it a win-win situation for everyone.

### 5.4.1 Belongs to

In general, you should know that a belongs-to association is meant to help you quickly retrieve an associated resource by defining a one-to-something association between two models (specifically, a child and its parent), where the child class should store its parent's key as a property. So, for instance, the following associates a comment with a user:

```
class Comment
  include DataMapper::Resource

  property :id, Serial
  property :body, Text

  belongs_to :user
end
```

Note that upon automigration the belongs-to association automatically creates the column `user_id` within the comments table. This means that any model object now has two new methods accessible, `user_id` and `user`. The first is simply the ID of the associated user, but the second actually retrieves the user resource for you. These defaults may not always fit your domain logic, however, so they can be altered.

```
class Paper
  include DataMapper::Resource

  property :id, Serial
  property :body, Text

  belongs_to :author, :class_name => "User",
    :child_key => [:author_id]

  belongs_to :reviewer, :class_name => "User",
    :child_key => [:reviewer_id]
end
```

Here we have two user objects parenting our paper object. To handle ambiguity, we use the hash keys `class_name` and `child_key`. The first is a string representation of the parent class name, and the second is an array indicating how the key should be stored within our child. At the end this produces the methods `author_id`, `reviewer_id`,

author, and reviewer on papers, where the first two are essentially association properties and the second two are means of retrieving the associated objects.

Let's take a look at how the `belongs_to` magic is performed:

```

module DataMapper::Associations
  def belongs_to(name, options={})
    @_valid_relations = false

    if options.key?(:class_name) &&
      !options.key?(:child_key)

      warn "... " # must set both
    end

    relationship = ManyToOne.setup(name, self, options)
  end
end

module DataMapper::Associations::ManyToOne

  # Set up many-to-one relationship between two models
  # -
  # @api private
  def self.setup(name, model, options = {})
    assert_kind_of 'name', name, Symbol
    assert_kind_of 'model', model, Model
    assert_kind_of 'options', options, Hash

    repository_name = model.repository.name

    model.class_eval <<-EOS, __FILE__, __LINE__
      def #{name}
        #{name}_association.nil? ?
          nil : #{name}_association
      end

      def #{name}=(parent)
        #{name}_association.replace(parent)
      end

      private

      def #{name}_association
        @@#{name}_association ||= begin
          unless relationship = model.relationships(
            #{repository_name.inspect})[:#{name}]

```

```

        raise ArgumentError,
          "Relationship #{name.inspect} "+
          "does not exist in \#{model}"
      end

      association = Proxy.new(relationship, self)
      child_associations << association
      association
    end
  end
EOS

model.relationships(repository_name)[name] =
  Relationship.new(
    name,
    repository_name,
    model,
    options.fetch(:class_name,
      Extlib::Inflection.classify(name)),
    options
  )
end
end

```

Starting with the `Associations` module, we can see that `belongs_to` fires off the creation of a many-to-one association. Moving on to `ManyToOne.setup`, we find extensive class evaluation where new methods for the association are defined. These allow us to get or set the association. Note that the `reader` method essentially proxies to the parent model (using a `Proxy` class later defined within `ManyToOne`). It also employs the `Relationship` class, `Mapper`'s most generalized way of storing information on associations within model classes. Finally, note that the use of `ManyToOne` does not strictly indicate that the relationship between the two models needs to be many-to-one. It may indeed be one-to-one (as determined within the other model), but from the perspective of the child model the more generalized `ManyToOne` class is appropriate for handling both possibilities.

### 5.4.2 Has

At this point you may be wondering about the flip side of the relationship, that is, the parent. `Has` associations are meant to handle this. The characteristics of `has` associations, however, differ in that they are meant to associate varying numbers of related model resources without storing information within the model object itself.

Let's create the counterpoint of the comment model we created in the previous section:

```
class User
  include DataMapper::Resource

  property :id, Serial
  property :login, String, :nullable => false

  has n, :comments
end
```

Note that the `has` method takes a minimum of two parameters. The first of these is the cardinality, which may be specified by a number, series, or `n`, and the second is the symbolized name of the associated class. If you're scratching your head over `n`, just know that it is equivalent to `1..0/0` and that it allows an indefinite number of associates. If you're coming from the ActiveRecord world, you can think of this as the "many" in `has_many`.

As we did before, we can tweak our relationship for the sake of the domain logic:

```
class User
  include DataMapper::Resource

  property :id, Serial
  property :login, String, :nullable => false

  has 1, :authored_papers, :class_name => "Paper",
    :child_key => [:author_id],
    :remote_name => :author

  has n, :reviewed_papers, :class_name => "Paper",
    :child_key => [:reviewer_id],
    :remote_name => :reviewer

end
```

There are plenty of things to notice this time. Once again, we've specified the associated class name along with the `child_key`, but we've also set `remote_name`, which is the symbolized name of the relationship in our other model. Last, note that we set the cardinality of the first to 1, which limits users to authoring only one paper, effectively making the relationship one-to-one.

Having now seen the use of `has`, let's go into the source to understand how it works:

```
module DataMapper::Associations
  def has(cardinality, name, options = {})
    # ...
  end
end
```



```

if name.kind_of?(Hash)
  name_through, through =
    name.keys.first, name.values.first
end

options = options.merge(
  extract_min_max(cardinality))
options = options.merge(
  extract_throughness(name))

# ... some warnings

klass = options[:max] == 1 ? OneToOne : OneToMany

# ... we'll show you later

relationship = klass.setup(
  options.delete(:name), self, options)
end

private

def extract_min_max(constraints)
  assert_kind_of 'constraints', constraints,
    Integer, Range unless constraints == n

  case constraints
  when Integer
    { :min => constraints, :max => constraints }
  when Range
    if constraints.first > constraints.last
      raise ArgumentError, "... "
    end
    { :min => constraints.first,
      :max => constraints.last }
  when n
    { :min => 0, :max => n }
  end
end

end

```

From this we can see that `has_many`, like `belongs_to`, creates an association, but that it may be `OneToOne` or `OneToMany` based upon the max cardinality. Because we've already looked inside one of these associations and because the others are set up in similar ways, we'll leave it up to you to explore further if you like.

### 5.4.3 Has through

You may need to work with one-to-many-through or many-to-many relationships. To handle these, DataMapper uses `through`. Let's tackle one-to-many-through first and then take a look at many-to-many relationships:

```
class Post
  include DataMapper::Resource

  has n, :taggings
  has n, :tags, :through => :taggings
end

class Tagging
  include DataMapper::Resource

  belongs_to :post
  belongs_to :tag
end

class Tag
  include DataMapper::Resource

  property :id, Serial
  property :value, String

  has n, :taggings
  has n, :posts, :through => :taggings
end
```

These examples show us three associated models where the `Tagging` class acts like a join table bridging the one-to-many relationships from both sides. Sometimes, though, you don't want to explicitly define this middle table. DataMapper lets you do this by setting `through` to `Resource`:

```
class Post
  include DataMapper::Resource

  has n, :post
  has n, :tags, :through => Resource
end

class Tag
  include DataMapper::Resource

  property :id, Serial
  property :value, String
```

```

  has n, :posts, :through => Resource
end

```

This automatically creates the bridging model for us dynamically. But this isn't magic; remember the line from the `def has` snippet of source code that we didn't show you? Here it is:

```

class = ManyToMany if options[:through] ==
  DataMapper::Resource

```

With that inside the previous snippet, it's easy to see that the use of the `through` option with `Resource` changes the association setup to a `ManyToMany`. This special association is used to create a join table model for you. Here's part of the setup method showing just that:

```

module DataMapper::Associations::ManyToMany

  def self.setup
    # ... the usual

    unless Object.const_defined?(model_name)
      model = DataMapper::Model.new(storage_name)

      model.class_eval <<-EOS, __FILE__, __LINE__
        def self.name; #{model_name.inspect} end
        def self.default_repository_name
          #{repository_name.inspect}
        end
        def self.many_to_many; true end
      EOS

      names.each do |n|
        model.belongs_to(
          Extlib::Inflection.underscore(n).gsub(
            '/', '_'
          ).to_sym)
      end

      Object.const_set(model_name, model)
    end

    relationship
  end
end

```

Note the particularly unique creation of a model through `Model.new` as opposed to a standard class definition. This is meant only for dynamically defined models like the one above.

## 5.5 CRUD basics

Mapper as an ORM is intended to create, retrieve, update, and delete records from a repository through interactions with Ruby objects. This means that we don't have to write SQL statements through the normal course of usage. In fact, Mapper's versatility, intelligence, and performance will probably leave you never needing to write a single SQL statement in your entire application.

Throughout this section, we will assume the existence of the following model:

```
class BlogEntry
  include Mapper::Resource

  property :id, Serial
  property :live, TrueClass
  property :title, String
  property :text, Text
end
```

### 5.5.1 Creating records

The creation of Mapper records is a two-step process. The first step is the creation of a new model object. This is as simple as initializing with the `new` method. This method can also take an attributes hash that will set the model object's properties. The second step of this process is the saving of the object's data into the database as a record. This is done via the `save` method. Below we create a new blog entry and then save it immediately after.

```
blog_entry = BlogEntry.new(:title => "Model Magic!",
  :text => "Persistently cool.")

blog_entry.save
```

At the end, this issues a SQL insert command, saving the data in our database. However, let's take a look first at the most superficial methods inside the Mapper that make this work:

```
module Mapper::Resource
  def save

    # ... association related

    saved = new_record? ? create : update

    if saved
      original_values.clear
```

```

    end

    # ... association related

    (saved | associations_saved) == true
end

def new_record?
  !defined?(@new_record) || @new_record
end

protected

def create
  return false if new_record? &&
    !dirty? && !model.key.any? { |p| p.serial? }

  # set defaults for new resource
  properties.each do |property|
    next if attribute_loaded?(property.name)
    property.set(self, property.default_for(self))
  end

  return false unless repository.create([ self ]) == 1

  @repository = repository
  @new_record = false

  # ... IdentityMap related

  true
end

private

def initialize(attributes = {})
  assert_valid_model
  self.attributes = attributes
end

end

```

As you can see, the default `initialize` has been overridden so that it can set attributes. You'll also spot a method `assert_valid_model`, but it isn't of much interest since all it does is confirm that the model class does in fact have properties defined. Moving on to the `save` method, you'll find that it first checks to see if the model object

should be a new record. To do this it uses the public method `new_record?`, which is also available to you should you need it as an application or plugin developer. Then, given that our record is new, it invokes the protected method `create`. This method effectively cascades through the resource's repository object down to an adapter, where a SQL create statement is executed.

Alternatively, you can shorten this process to a single step by using the class method `create` defined within the `Model` module. Here we use it just as we did before:

```
blog_entry = BlogEntry.create(:title => 'Models Rule!',
  :text => 'Persistently cool.')
```

Taking a peek at the source code, we find that the class method `create` does exactly what we did ourselves before but returns the model object for our convenience:

```
module DataMapper::Model

  def create(attributes = {})
    resource = new(attributes)
    resource.save
    resource
  end

end
```

## 5.5.2 Retrieving records

The retrieval of model records is principally done through the two methods `all` and `first`. These two methods pull up a collection of records or access a single record, respectively. They can easily be chained, allowing for the refining of the data to be retrieved. Let's take a look at some basic examples:

```
user = User.first(:login => 'foysavas')
groups = Group.all(:name => '%Ruby%')
admin_groups = groups.all(:user => user)
```

The first line looks up a user by login, the second retrieves all groups with the word *Ruby* in them, and the third refines the collection of the second to only those where the user is the admin. Let's look at the source behind the methods `first` and `all` to get an understanding of how they work:

```
module DataMapper::Model

  def all(query = {})
    query = scoped_query(query)
    query.repository.read_many(query)
  end
```

**end**

```
def first(*args)
  query = args.last.respond_to?(:merge) ?
    args.pop : {}

  query = scoped_query(
    query.merge(:limit => args.first || 1))

  if args.any?
    query.repository.read_many(query)
  else
    query.repository.read_one(query)
  end
end
```

**end**

Both `all` and `first` use the method `scoped_query` to integrate new query parameters with any preexisting ones that may exist higher up on a collection on which the method may be acting:

```
module DataMapper::Model

  private

  def scoped_query(query = self.query)
    assert_kind_of 'query', query, Query, Hash

    return self.query if query == self.query

    query = if query.kind_of?(Hash)
      Query.new(query.has_key?(:repository) ?
        query.delete(:repository) :
        self.repository, self, query)
    else
      query
    end

    if self.query
      self.query.merge(query)
    else
      merge_with_default_scope(query)
    end
  end

end
```

DataMapper uses the method `assert_kind_of` as a way of enforcing types and throws errors when types do not match. Thus, above, we see that `scoped_query` accepts only queries and hashes. The hashes are really just cases of yet-to-be-initiated queries coming from the parameters of some method like `all` or `first`. If both new query parameters and an existing query exist, the two are merged. The model's default scope (typically having no conditions and all non-lazy model fields) is used to merge in further conditions.

### Design decision: query object algebra

If there was ever something to be excited about with regard to DataMapper, this is it. Under the hood, the reason why DataMapper can do things like chain retrieval methods and use strategic eager loading is that it essentially handles queries as elements of an algebraic structure. This is completely unlike most ORMs, which simply treat queries as undesirable SQL remnants to be executed. We recommend letting this subtlety take its time to soak in. However, the essence of it all is that with each call of a retrieve method, DataMapper creates a new query by operating on two other queries (one of them possibly empty). With this fresh perspective on DataMapper, we recommend visiting the code behind the method `Query#update`, which we're just about to do.

The method `Query#merge` duplicates the query and then seeks to update it. Below we see this method as it leads into `Query#update`.

```
class DataMapper::Query

  def update(other)
    assert_kind_of 'other', other, self.class, Hash

    assert_valid_other(other)

    if other.kind_of?(Hash)
      return self if other.empty?
      other = self.class.new(@repository, model, other)
    end

    return self if self == other

    @reload = other.reload?
    unless other.reload? == false
    @unique = other.unique?
    unless other.unique? == false
    @offset = other.offset
```



```

    if other.reload? || other.offset != 0
      @limit = other.limit
      unless other.limit == nil
        @order = other.order
        unless other.order == model.default_order
          @add_reversed = other.add_reversed?
          unless other.add_reversed? == false
            @fields = other.fields
            unless other.fields == @properties.defaults
              @links = other.links
              unless other.links == []
                @includes = other.includes
                unless other.includes == []

                  update_conditions(other)

                self
              end

            def merge(other)
              dup.update(other)
            end

          end
        end
      end

```

Note that the method `update` picks out special query parameters before updating the conditions and finally returning itself.

### 5.5.2.1 Special query parameters

The parameters passed into `all` and `first` are mostly understood simply as conditions upon parameters. However, certain keys are understood as special query parameters that shape the query in other ways. The following list should make the use of each of these clear:

- `add_reversed`—reverses the order in which objects are added to the collection. Defaults to `false`.
- `conditions`—allows SQL conditions to be set directly using an array of strings. Conditions are appended to conditions specified elsewhere.
- `fields`—sets the fields to fetch as an array of symbols or properties. Defaults to all of a model's non-lazy properties.
- `includes`—includes other model data specified as a list of `DataMapper` property paths.

- `limit`—limits the number of records returned. Defaults to 1 in the case of `first` and is otherwise not set.
- `links`—links in related model data specified by an array of symbols, strings, or associations.
- `offset`—the offset of the query, essential for paging. Defaults to 0.
- `order`—the query order specified as an array or properties (or symbols) modified by the two direction methods `desc` and `asc`.
- `reload`—causes the reloading of the entire data set. Defaults to false.
- `unique`—groups by the fields specified, resulting in a unique collection. Defaults to false.

### 5.5.2.2 Lazy loading of collections

`DataMapper` does not load collections or issue database queries until the data is absolutely needed. The major benefit here is that application developers can worry less about the database side of things once again, knowing that unless they actually use the data of a resource, no database query will be executed. With Merb, we've also found that this means simpler controller code, since we can use chained relationships or pagination inside the view. With any other ORM, this may be extremely bad form, given that it implies littering the view with lines of supporting code as well as incurring performance penalties based on the retrieval of possibly unused data. Below we present the practical application of collection lazy loading.

```
# Posts Controller
# app/controllers/posts.rb

class Posts
  before :set_page

  def index
    @posts = Post.all
    render
  end

  private

  def set_page
    @p = params[:page] > 0 ?
      params[:page] : 1
  end
end
```

```
# Posts Index View
# app/views/posts/index.html.haml
- @posts.all(:limit => 10, :offset => 10*@p).each do |i|
  .post
    = @posts.name
```

Note that this executes only one database query, specifically at the each. To see how and why this works, we need to take a look at some of the code in the parent class of Collection, LazyArray:

```
class LazyArray # borrowed partially from StrokeDB
  instance_methods.each { |m|
    undef_method m unless %w[
      __id__ __send__ send class dup object_id
      kind_of? respond_to? equal? assert_kind_of
      should should_not instance_variable_set
      instance_variable_get extend ].include?(m.to_s)
    }

  # add proxies for all Array and Enumerable methods
  ((Array.instance_methods(false)
   | Enumerable.instance_methods(false)).map { |m|
    m.to_s
  } - %w[ taguri= ]).each do |method|

    class_eval <<-EOS, __FILE__, __LINE__
      def #{method}(*args, &block)
        lazy_load
        results = @array.#{method}(*args, &block)
        results.equal?(@array) ? self : results
      end
    EOS

  end

  def load_with(&block)
    @load_with_proc = block
    self
  end

  # ...

  private

  def lazy_load
    return if loaded?
```

```

    mark_loaded
    @load_with_proc[self]
    @array.unshift(*@head)
    @array.concat(@tail)
    @head = @tail = nil
    @reapers.each { |r|
      @array.delete_if(&r)
    } if @reapers
    @array.freeze if frozen?
end

# ...

end

```

Starting at the top, we can see that all but the quintessence methods are undefined. This is because `LazyArray` is meant to emulate the primitive `Array` class, and starting off with a slate that is as blank as possible helps us get there. The next few lines define various instance methods from both `Array` and `Enumerable`, essentially making `LazyArray` a proxy to a real array but prefacing the call of any array method with `lazy_load`. The `lazy_load` method itself either simply returns true if already loaded, or uses a Proc defined through the `load_with` method to populate the array. All in all, the lazy loading of `LazyArray` has a profound impact on the `DataMapper` API, arguably serving as the foundation for the elegance and straightforwardness of the query algebra.

### 5.5.2.3 Lazy loading of properties

Some property data is not automatically retrieved when a model object is loaded. For instance, by default, text properties are not loaded unless you specifically request them. This form of lazy loading is facilitated by code with the `Resource` module and `PropertySet` class. Let's see it in action before taking an in-depth look at how it has been put together:

```

# app/models/post.rb
class Post
  include DataMapper::Resource

  property :id, Serial
  property :title, String
  property :body, Text
end

# Example Merb Interaction
> post = Post.first

```

```

~ SELECT "id", "title", "is_basic" FROM "posts" ORDER BY
  "id" LIMIT 1
=> #<Post id=1 title="First Post!" body=<not loaded>>
> post.body
~ SELECT "body", "id" FROM "posts" WHERE ("id" = 1) ORDER BY "id"
=> "Nothing to see here"

```

Note that if we had multiple text properties, they would all have been loaded by the second line of interaction. To prevent this from happening, you can define lazy contexts on properties, thus segmenting the retrieval of lazy property data:

```

# app/models/post.rb
class Article
  include DataMapper::Resource

  property :id, Serial
  property :title, String, :lazy => true
  property :abstract, Text, :lazy => [:summary, :full]
  property :body, Text, :lazy => [:full]
end

```

It's time to see how this is done. We'll have to open up `Resource` and `PropertySet`, with the insight that a property when either get or set calls the method `Resource#lazy_load`:

```

module DataMapper::Resource

  def lazy_load(name)
    reload_attributes(
      *properties.lazy_load_context(name) -
      loaded_attributes)
  end

end

class DataMapper::PropertySet

  # ...

  def property_contexts(name)
    contexts = []
    lazy_contexts.each do |context, property_names|
      contexts << context
      if property_names.include?(name)
    end
    contexts
  end

```

```

def lazy_load_context(names)
  if names.kind_of?(Array) && names.empty?
    raise ArgumentError, '+names+ cannot be empty',
      caller
  end

  result = []

  Array(names).each do |name|
    contexts = property_contexts(name)
    if contexts.empty?
      result << name # not lazy
    else
      result |= lazy_contexts.values_at(*contexts).
        flatten.uniq
    end
  end
  result
end
end

```

The methods of `PropertySet` aren't anything special, but seeing how they work certainly clears up any ambiguity that may have existed within the concept of lazy load contexts.

#### 5.5.2.4 Strategic eager loading

If you've used `ActiveRecord` before, you've probably trained yourself to avoid  $N+1$  queries. These come up frequently in `ActiveRecord` since iteration over the associates of a model object usually forces you to make a query for each associate. Add in the original query for the model itself, and you have  $N+1$  queries in total. However, `DataMapper` prevents this from happening and instead issues only two queries. Let's take a look at an example in a view to make this more concrete:

```

<% Post.all.each do |post| %>
  <div class="post">
    <h1><%= post.title %></h1>
    <h2>by <%= post.author.name %></h2>
  </div>
<% end %>

```

With the code above, all posts and the names of their authors are outputted using only two queries: the first to get the posts and the second to get their authors. This kind of elimination of  $N+1$  queries is called **strategic eager loading** and is possible thanks to a combination of many different `DataMapper` implementation decisions. To get an

idea of how strategic eager loading works, let's take a look at some code inside the `Relationship` class that would have been used in the previous example:

```
class DataMapper::Associations::Relationship

  # ...

  # @api private
  def get_parent(child, parent = nil)
    child_value = child_key.get(child)
    return nil if child_value.any? { |v| v.nil? }

    with_repository(parent || parent_model) do
      parent_identity_map = (parent || parent_model).
        repository.identity_map(parent_model.base_model)
      child_identity_map = child.
        repository.identity_map(child_model.base_model)

      if parent = parent_identity_map[child_value]
        return parent
      end

      children = child_identity_map.values
      children << child
      unless child_identity_map[child.key]

        bind_values = children.map {
          |c| child_key.get(c) }.uniq
        query_values = bind_values.reject {
          |k| parent_identity_map[k] }

        bind_values = query_values
        unless query_values.empty?
          query = parent_key.zip(bind_values.transpose).
            to_hash
          association_accessor =
            "#{self.name}_association"

          collection = parent_model.send(:all, query)
          unless collection.empty?
            collection.send(:lazy_load)
            children.each do |c|
              c.send(association_accessor).
                instance_variable_set(
                  :@parent,
                  collection.get(*child_key.get(c)))
            end
          end
        end
      end
    end
  end
```

```

        child.send(association_accessor).
            instance_variable_get(:@parent)
    end
end
end

end

```

From this we learn that in the process of getting a parent resource, `Mapper` pulls up the identity map of the parent model and child model to see if the resource has already been loaded. If it has, `Mapper` short-circuits any retrieval and simply returns the appropriate parent. Most important, if the resource is not already loaded, `Mapper` uses the parent keys from all the relevant children within a collection query. The results are then loaded immediately, and after all children are connected with their parents, the parent requested is returned.

### 5.5.3 Updating records

Resources can be updated by using the `save` method similarly to how it was used with record creation. However, for saving to have any effect, it is necessary that at least one property value be recently set. This causes `Mapper` to mark certain properties as dirty and use them during the creation of an update statement. Below we display the two ways of setting a property value and causing it to be marked as dirty.

```

post = Post.first
post.title = "New Title"
post.attribute_set(:body, "New Body")
post.save

```

However, note that the second method `attribute_set` is typically reserved for use inside override writer methods. Note that `save`, in the case of non-new records, cascades to the calling of `update`. Thus we have the option of using that method directly if we want:

```

post.update

```

#### 5.5.3.1 Using `update_attributes`

There is one other way to invoke the updating of attributes. This is to use the method `update_attributes`, which accepts an arbitrary hash and then an optional constraining property array. Consequently, it works well with form parameters a user may have passed in:

```

class Users
  def update

```



```

if @user.update_attributes(params[:user],
  [:name, :email, :description])
  redirect resource(@user)
else
  render :edit
end
end
end

```

Here we have constrained the user to being able to update only name, email, and description.

### 5.5.3.2 Original and dirty attribute values

You may at some point want to enhance model logic through the comparison of original and dirty attribute values. Here we do so within the method `update_speed`:

```

class Position
  property :id, Serial
  property :vertical_position, Integer
  property :horizontal_position, Integer
  property :speed, Float

  belongs_to :player

  before :save, :update_speed

  private

  def update_speed
    dy = 0
    dx = 0
    if original_values[:vertical_position]
      dy = vertical_position -
        original_values[:vertical_position]
    end
    if original_values[:horizontal_position]
      dx = horizontal_position -
        original_values[:horizontal_position]
    end
    v = Math.sqrt(dx*dx + dy*dy)
    attribute_set(:speed, v)
  end
end

```

You may notice that we use a `before` hook here. We'll cover hooks in the next section.

### 5.5.4 Destroying records

Records can be deleted by using the `destroy` method. Alternatively, if you're looking to delete a full collection of resources, you can use the method `destroy!`:

```
User.first(:id => 2).destroy
Post.all(:user_id => 2).destroy!
```

## 5.6 Hooks

DataMapper hooks differ from those used in Merb controllers. This is because controller filter chains require a decently specific form of logic. DataMapper models, on the other hand, have the benefit of using the more generic `Hook` class of `Extlib`.

### Design decision: explicit hooks over aliasing of methods

DataMapper could have left it up to application developers to fend for themselves when it comes to hooks. After all, Ruby is an extremely versatile language, and the `alias` method could be used to chain model methods like `create` and `save`. However, DataMapper developers, like Merb developers, shared a distaste for the aliased decoration of methods for, among other reasons, its confusing implications on stacktraces. Therefore, `Extlib::Hook` was created to enable application developers to easily decorate both instance and class methods without doing any aliasing.

From the application developer's perspective, the two methods used to create hooks are `before` and `after`. DataMapper registers specific hooks for the methods `save`, `create`, `update`, and `destroy`. Each of these can be used with `before` and `after`.

```
module DataMapper
  module Hook
    def self.included(model)
      model.class_eval <<-EOS, __FILE__, __LINE__
        include Extlib::Hook
        register_instance_hooks :save, :create,
                               :update, :destroy
      EOS
    end
  end
  DataMapper::Resource.append_inclusions Hook
end # module DataMapper
```

## 5.7 Plugins

Modularity has been a central objective of `DataMapper`. Thus, many of the features that you might otherwise expect within a standard ORM are with `DataMapper` found as plugins. This includes timestamping, aggregation, validations, and various data structures. In this section we'll go over the most fundamental of these plugins, understanding not only how they're used but also how they work.

### 5.7.1 Extra property types

The package `dm-types` provides numerous additional property types. Here's a list of those included:

- `BCryptHash`—encrypts a string using the `bcrypt` library
- `Csv`—parses strings as CSVs using `FasterCSV`
- `Enum`—stores an enumerated value as an integer
- `EpochTime`—converts `Time` and `DateTime` to `EpochTime`, that is, the number of seconds since the beginning of UNIX time
- `FilePath`—stores paths as strings using `Pathname`
- `Flag`—binary flags stored as integers
- `IpAddress`—IP address stored as a string
- `Json`—JSON stored as a string
- `Regexp`—regular expressions stored as strings
- `Serial`—an auto-incrementing integer type
- `Slug`—escapes a stored string, making it suitable to be used as part of a URL
- `URI`—stores an `Addressable::URI` as a string
- `UUID`—creates a UUID stored as a string
- `Yaml`—stores YAML as a string

Let's take a look at the source to one of these for a better understanding of how to create our own types:

```
require 'yaml'
```

```
module DataMapper
  module Types
    class Yaml < DataMapper::Type
      primitive String
    end
  end
end
```

```

size 65535
lazy true

def self.load(value, property)
  if value.nil?
    nil
  elsif value.is_a?(String)
    ::YAML.load(value)
  else
    raise ArgumentError.new(
      "+value+ must be nil or a String")
  end
end

def self.dump(value, property)
  if value.nil?
    nil
  elsif value.is_a?(String) && value =~ /^---/
    value
  else
    ::YAML.dump(value)
  end
end

def self.typecast(value, property)
  # Leave values exactly as they're provided.
  value
end
end # class Yaml
end # module Types
end # module DataMapper

```

As you can see, new `DataMapper` types can be created by subclass off of `DataMapper::Type`. You will then have to set the primitive type, and this can be done using the class method `primitive`. You may additionally have to set attributes like `size` and `laziness` as was done in the case above. Finally, the two methods that do the hard work are the class methods `load` and `dump`. These need to be defined only if the custom type needs to override them from simply returning the value. With the `Yaml` type, strings are converted into YAML when loaded from the database and are converted to strings when they need to be dumped into the database.

## 5.7.2 Timestamps

The gem `dm-timestamps` is one of the most commonly used `DataMapper` plugins. It saves you from having to code timestamping into your models. Note that once the gem is included, it applies to all `DataMapper` models. Thus we can set the following

four properties in any Merb stack model, knowing they will automatically be set when needed:

```
class User
  property :created_at, DateTime
  property :created_on, Date
  property :updated_at, DateTime
  property :updated_on, Date
end
```

Because `dm-timestamps` is a decently simple plugin but also reveals the foundation of resource extension plugins, let's take a quick look:

```
module DataMapper
  module Timestamp
    Resource.append_inclusions self

    TIMESTAMP_PROPERTIES = {
      :updated_at => [ DateTime,
        lambda { |r, p| DateTime.now } ],

      :updated_on => [ Date,
        lambda { |r, p| Date.today } ],

      :created_at => [ DateTime,
        lambda { |r, p|
          r.created_at ||
            (DateTime.now if r.new_record?) } ],

      :created_on => [ Date,
        lambda { |r, p|
          r.created_on ||
            (Date.today if r.new_record?) } ],
    }.freeze

    def self.included(model)
      model.before :create, :set_timestamps
      model.before :update, :set_timestamps
      model.extend ClassMethods
    end

    private

    def set_timestamps
      return unless dirty?
      TIMESTAMP_PROPERTIES.each do |name, (_type, proc)|
        if model.properties.has_property?(name)
          model.properties[name].set(self,
            proc.call(self,
```

```

        model.properties[name])) unless
          attribute_dirty?(name)
        end
      end
    end

module ClassMethods
  def timestamps(*names)
    raise ArgumentError, '...' if names.empty?

    names.each do |name|
      case name
      when *TIMESTAMP_PROPERTIES.keys
        type, proc = TIMESTAMP_PROPERTIES[name]
        property name, type
      when :at
        timestamps(:created_at, :updated_at)
      when :on
        timestamps(:created_on, :updated_on)
      else
        raise InvalidTimestampName,
          "Invalid timestamp property name '#{name}'"
      end
    end
  end
end # module ClassMethods

class InvalidTimestampName < RuntimeError; end
end # module Timestamp
end # module DataMapper

```

The first line to notice is the third one. Here the module appends itself to `Resource`. As we saw earlier in this chapter, this gets the `Timestamp` module automatically included in all classes that include `DataMapper::Resource`. Moving on, we see the definition of a number of lambdas to be used in setting the four basic timestamps. This is followed by the class method `included`, which sets up before hooks to apply the timestamps. It all extends our model classes with a `timestamps` class method. This is a convenience method for defining the various timestamp properties tersely.

### 5.7.3 Aggregates

The `DataMapper` core has been designed to limit its use as a reporting tool and simply act as an ORM. The plugin `dm-aggregates` consequently adds in some of the most common aggregating methods used by SQL databases:

- `count`—finds the number of records in a collection by directly using a count SQL statement and not the Ruby `size` method
- `min`—finds the minimum value of a numerical property using SQL
- `max`—finds the maximum value of a numerical property using SQL
- `avg`—finds the average value of a numerical property using SQL
- `sum`—totals the values of a numerical property using SQL

Chances are you will use `dm-aggregates` at some point. However, before we look into the source, it's best to recognize that the plugin essentially extends `DataMapper`'s capability to what it was not really meant to do.

```
module DataMapper
  class Collection
    include AggregateFunctions

    private

    def property_by_name(property_name)
      properties[property_name]
    end
  end
end

module Model
  include AggregateFunctions

  private

  def property_by_name(property_name)
    properties(repository.name)[property_name]
  end
end

module AggregateFunctions
  def count(*args)
    query = args.last.kind_of?(Hash) ? args.pop : {}
    property_name = args.first

    if property_name
      assert_kind_of 'property',
        property_by_name(property_name), Property
    end
  end
end
```

```

    aggregate(query.merge(:fields =>
      [ property_name ?
        property_name.count : :all.count ]))
  end
end

module Adapters
  class DataObjectsAdapter
    def aggregate(query)
      with_reader(read_statement(query),
        query.bind_values) do |reader|

        results = []

        while(reader.next!) do
          row = query.fields.zip(
            reader.values).map do |field,value|

            if field.respond_to?(:operator)
              send(field.operator, field.target, value)
            else
              field.typecast(value)
            end
          end

          results << (query.fields.size > 1 ?
            row : row[0])
        end

        results
      end
    end

    private

    def count(property, value)
      value.to_i
    end

    module SQL
      private

      alias original_property_to_column_name
      property_to_column_name

      def property_to_column_name(repository,
        property, qualify)

```



```

case property
  when Query::Operator
    aggregate_field_statement(repository,
      property.operator, property.target, qualify)

  when Property, Query::Path
    original_property_to_column_name(repository,
      property, qualify)

  else
    raise ArgumentError, "..."
  end
end

def aggregate_field_statement(repository,
  aggregate_function, property, qualify)

  column_name = if aggregate_function == :count
    && property == :all
    '*'
  else
    property_to_column_name(repository, property, qualify)
  end

  function_name = case aggregate_function
    when :count then 'COUNT'
    # ...
    else raise "Invalid ... "
  end

  "#{function_name}({column_name})"
end
end # module SQL

include SQL

end
end

```

Above we have included only the code covering the `count` method. However, it's easy to recognize the substantial monkey patching going on, particularly in the case of the `SQL` methods. Otherwise, though, this is a great example of the trickling down of method calls from collections and models into the adapter where `SQL` statements are formed.

### 5.7.4 Validations

The plugin `dm-validations` validates the property values of model objects before saving them. This means that if a model object returns false upon save, you can most likely interpret it as having been caused by undesirable values on properties. Another way to check if a particular model is valid is to directly use the `valid?` method that is squeezed in before `create` or `update`. Before we go any further, here's a list of the validation methods available within your models through `dm-validations`:

- `validates_present`—validates the presence of an attribute value.
- `validates_absent`—validates the absence of an attribute value.
- `validates_is_accepted`—validates that an attribute is true or optionally not false through `:allow_nil => true`. It can also work with a custom set of acceptance values using `:accept => [values]`.
- `validates_is_confirmed`—validates the confirmation of an attribute with another attribute, for instance, matching `password` and `password_confirmation`. The default confirmation attribute is the original attribute ending in `_confirmation`, but `:confirm` can be used to set it to anything else.
- `validates_format`—validates the format of an attribute value against a regular expression or Proc associated by `:with`. Alternatively, it can be used with predefined formats such as `Email` and `Url` through `:as`. The `:allow_nil` key is also available.
- `validates_length`—validates the value of a numeric against a `:min` or `:max` value. Alternatively, a range can be used along with `:within`.
- `validates_with_method`—validates either the model as a whole or a specific property through a method. If only one parameter is given, it is the symbolic form of the method to check the entire model. If two are given, they are the attribute and the method used to check that attribute. Error messages can be passed as true by returning an array where false is the first element and a string for an error message is the second from the validating method.
- `validates_with_block`—like `validates_with_method` but uses blocks. You can validate either against the whole model or a specific attribute as well as pass in error messages.
- `validates_is_number`—validates that the value of an attribute is a number, appropriate for use in checking the precision and scale of floats.
- `validates_is_unique`—validates that the attribute value is unique, either within the scope of the attribute value of all other model objects or some other scope specified

by an array of property symbols through `:scope`. Also accepts `:allow_nil` to be set.

- `validates_within`—validates that an attribute value is within a set of values specified by `:set`.

Alternatively, instead of directly using the validations methods, you can include particular hash key-and-values on property definitions. Here's a list of what keys automatically create appropriate validations:

- `:nullable`—when set to false, automatically creates a presence validator
- `:length` or `:size`—automatically creates a length validator
- `:format`—creates a format validator
- `:set`—creates a within validator

Additionally, numerical properties are automatically validated using `validates_is_number`. To turn off autovalidation on this or any other property type, use `:auto_validation => false`.

#### 5.7.4.1 Conditions

Validation methods are also capable of generally accepting conditions as Procs assigned to `:if` or `:unless`. The single block parameter for these Procs is the resource itself. Thus we can do the following:

```
class Experiment
  include DataMapper::Resource

  property :id, Serial
  property :name, String
  property :impetus, Text
  property :question, Text
  property :hypothesis, Text
  property :description, Text
  property :conclusion, Text
  property :completed, TrueClass
  property :result, TrueClass

  validates_present :conclusion, :if => proc { |r|
    r.completed? && r.result?
  }
end
```

For terseness, DataMapper validations also allow us to specify a method as a symbol instead of a full Proc. Here we require only that an experiment be complete for it to have a conclusion:

```
validates_present :conclusion, :if => :completed?
```

#### 5.7.4.2 Contexts

Contexts allow us to do validations with similar conditions, but specified at the point of validation. For instance, assuming we have the same Experiment model from before, we may set different contexts on particular property validations specifying that they must be validated together:

```
validates_present :impetus, :when => [:proposal]
validates_present :question, :when => [:proposal]
validates_absent :completed, :when => [:proposal]
```

The array assigned to **when** is an array of contexts. The default context is known as **:default**. We can now use these contexts by including them as a parameter with **valid?**:

```
exp = Experiment.new(
  :name => 'Great Subjective Experiment',
  :impetus => 'Thoughts on physicalism and the mind',
  :question => 'Is it possible to subjectively test ' +
    'the consciousness of other modes of thought ' +
    'through their integration with your own?'
  :completed => true)
exp.valid?(:proposal) # => false
```

#### 5.7.4.3 Errors

Every time a model object is validated, it populates (or empties) a hash of errors accessible through the resource instance method **errors**. These errors can be used to indicate to a user that something went wrong or otherwise recognize what particular attributes are invalid:

```
resource.errors.each do |e|
  puts e # => [[:attr_name, ["Error!"]]]
end

resource.errors.on(:attr_name) # => ["Error!"]
resource.errors.on(:another_attr) # => nil
```

Notice how **errors.on** returns nil when there are no errors. Consequently it can be used to test if an error is present on a property. Also note that error messages are

given as arrays. This is because multiple validation errors may have occurred on a single attribute.

## 5.8 Conclusion

DataMapper is an undeniably excellent ORM. It offers application developers the chance to stay as far away from database work as possible by streamlining development migrations and placing model properties within the model. It also lets us treat ORM objects even more casually by virtue of its eager and lazy loading of data sets. Inside the Merb stack or not, the highly modular DataMapper will serve you well.

# Index

---

- = (equal sign)
  - outputting results, 107–108
  - outputting strings, 108
  - sanitizing, 108
- # (pound sign)
  - comment indicator, 103
  - in ID names, 106
- / (slash)
  - in file paths, 21
  - path expansion, 22
- . (dot), in class names, 106
- & (ampersand), anchors in YAML nodes, 115
- == (equal signs), sanitizing, 108
- (hyphen), code indicator, 107
- % (percent sign), tag indicator, 105
- { } (curly braces), in Haml tags, 107
- ( ) (parentheses), in optional route matching, 52
- <% %> ERB delimiters, 102
- <% =%> ERB delimiters, 102
- <%= %> ERB delimiters, 102
- <%= -%> ERB delimiters, 102
- ~ (tilde), preserving whitespace, 108

## A

- `absolute_url` method, 71
- Abstract controllers, 69–70
- `abstract!` method, 229, 238–239

- `abstract?` method, 229
- `AbstractController` class, methods
  - `before`, 69
  - `absolute_url`, 71
  - `action_name`, 70
  - `action_name=`, 70
  - `add_filter`, 69
  - `after`, 69
  - `body`, 71
  - `body=`, 71
  - `capture`, 71
  - `capture_erb`, 71
  - `catch_content`, 71
  - `clear_content`, 71
  - `concat`, 71
  - `concat_erb`, 71
  - `content_type`, 71
  - `content_type=`, 71
  - `controller_name`, 69–70
  - `default_layout`, 70
  - `filter`, 69
  - general, 69, 70
  - `inherited`, 69
  - `instance`, 70
  - `layout`, 70
  - `normalize_filters!`, 69
  - `partial`, 71

- AbstractController** class,
  - methods (*continued*)
  - `relative_url`, 71
  - `render`, 70
  - `render`, 71
  - `render_all`, 71
  - `render_html`, 71
  - `render_js`, 71
  - `render_json`, 71
  - `render_options`, 70
  - `render_text`, 71
  - `render_xml`, 71
  - `render_yaml`, 71
  - `skip_after`, 69
  - `skip_before`, 69
  - `skip_filter`, 69
  - `subclasses_list`, 69
  - `template_roots`, 70, 71
  - `template_roots=`, 70, 71
  - `throw_content`, 71
  - `throw_content?`, 71
  - `URL`, 71
  - `url`, 71
  - `view`, 71
- Access privileges, properties, 131
- Access realms, 240
- `:accessor` option, 131
- Action caching, 274–275
- `action` key, 42
- `action_name` method, 70
- `action_name=` method, 70
- Actions
  - callable, returning, 72
  - callable, setting, 79
  - calling, 66
  - hiding/showing, 72, 79
  - mailers
    - invoking, 258–259
    - naming, 257–258
  - methods available as, 68
  - names, 70
  - parts, 266–267
  - readability, improving, 78
- ActionStore** strategy store, 270
- ActiveRecord**, definition, 26
- ActiveRecord** associations, *versus* **DataMapper**, 132–133
- Adapter options, Rack, 23–25
- Adapters, **DataMapper**, 114
- `add_filter` method, 69
- Adding
  - controller names to
    - `subclasses_list`, 69
  - CSS error class, 186
  - filters to filter chains, 69
  - names to routes, 57–58
  - query parameters, 143
  - routes, 44–45
  - session functionality, 29
- `add_rakefiles` plugin, 222
- Addressable URIs, storing as strings, 155
- `add_reversed` query parameter, 145
- `add_slice` method, 202–203
- AdhocStore** strategy store, 270
- `after` class method, 81–82
- After filter chain, 69
- After filters, 81–82
- `after` method
  - AbstractController** class, 69
  - authentication strategies, 229
  - creating hooks, 154
  - filter method, 69
  - Strategy** class, 229
- `after_app_loads` blocks, 30
- AfterAppLoads** boot loader, 30
- `after_app_loads` method, 13–14
- `after_authentication` method, 237
- `after_callbacks` array, 237
- Aggregates, 158–161

- `all` method, 142–145
- `alter_table_column_statement`
  - method, 126
- Ampersand (&), anchors in YAML nodes, 115
- `app` argument, 3
- `app/` directory, 9
- `Application` class, 31, 72
- Application controllers, 72–74
- Application layouts
  - alternative, specifying, 28–29
  - core, 3
  - custom, 3
  - flat, 2–3, 6–8
  - overview, 2–3
  - standard, 3, 8–9
  - very flat, 2–6
- Applications
  - creating, 1–2
  - development environment, 7, 16–17
  - environments, 7, 16–17
  - nontemplate code, loading, 29
  - production environment, 7, 16–17
  - reloading, 11
  - single file. *See* Flat application layouts; Very flat application layouts.
  - testing environment, 7, 16–17
- `assert_kind` method, 144
- Associations
  - belongs to, 132–135
  - `belongs_to` method, 132–135
  - cardinality, specifying, 136
  - `DataMapper` *versus* `ActiveRecord`, 132–133
  - has, 132–133, 135–137
  - `has` method, 132–133
  - has through, 138–139
  - join table models, creating, 139
  - `ManyToMany` associations, 139
  - many-to-many-through, 139
  - one-to-many-through, 138–139
  - overview, 132–133
  - symbolized class names, 136
  - through method, 138–139
- Attaching files to email, 260–261
- Attributes, updating, 152–153
- `attribute_set` method, 152
- Audit routes rake task, 48
- `audit:routes` command, 48
- `auth` core. *See* Authentication, `auth` core.
- `auth` more. *See* Authentication, `auth` more.
- `auth` password slices. *See* Authentication, `auth` password slices.
- `authenticate` method, 235
- `authenticate!` method, 225–226
- Authentication, `auth` core
  - `abstract!` method, 229
  - `abstract?` method, 229
  - `add_rakefiles` plugin, 222
  - `after` method, 229
  - `after_authentication` method, 237
  - `after_callbacks` array, 237
  - `authenticate` method, 235
  - `authenticate!` method, 225–226
- `Authentication` class, 223–227
- `before` method, 229
- `before_app_loads` plugin, 222
- callbacks, 236–237
- customizations, 236
- `customize_default` method, 236
- `ensure_authenticated` helper, 234–235
- error messages, storing, 232–233
- `halt!` method, 230
- `halted?` method, 230
- helpers, 234–235
- hooking methods, 224
- `inherited` method, 229
- `initialize` method, 230



- Authentication, `auth` core (*continued*)
    - model class, selecting, 223
    - `push_path` method, 222
    - `redirect` method, 230
    - responses, 233–234
    - `responses.rb` file, 233–234
    - router helper, 235
    - `run_after_authentication_`
      - `callbacks` method, 226–227, 237
    - serial callbacks, 226–227
    - sessions, 231
    - `setup.rb` file, 222
    - `strategies.rb` file, 222
    - `Strategy` class, 227–231
    - `user_class` method, 231
  - Authentication, `auth` more
    - controller, 245
    - `encrypt` method, 243
    - models, 242–245
    - overview, 237–238
    - passwords, encrypting, 243–244
    - `redirect_back_or` method, 245
    - redirecting logins, 245
    - strategies
      - `abstract!` method, 238–239
      - access realms, 240
      - `Base` class, 238
      - basic, 239–241
      - `basic_authentication?`
        - method, 240
      - `login_param` method, 239
      - OpenID, 242
      - password form, 241
      - `password_param` method, 239
      - `run!` method, 239
  - Authentication, `auth` password slices
    - controller, 247–249
    - `destroy` action, 248
    - `error_messages_for` method, 250
    - `Exceptions` controller, 247–249
    - `lib` files, 246–247
    - login view, 249–250
    - `Sessions` controller, 247–249
    - underscored filters, 248
    - `update` action, 248
    - views, 249–250
  - Authentication class, 223–227
  - Auto-increment, enabling, 131
  - Auto-incrementing integers, 155
  - Automatic
    - form actions, setting, 187
    - parameters, routing match rules, 51
    - parameters, symbolic matching, 51
    - validation, 132, 163
  - `auto_migrate!` method, 123–126
  - `auto_upgrade!` method, 123–126
  - `:auto_validation` option, 132
  - Average value, calculating, 159
  - `avg` method, 159
- ## B
- `Base` class, 238
  - `basic_authentication?` method, 240
  - `BCryptHash` property type, 155
  - `be_client_error` matcher, 291–292
  - `before` class method, 80–81
  - Before filter chain, 69
  - Before filters, 80–81
  - `before` method
    - `AbstractController` class, 69
    - authentication strategies, 229
    - creating hooks, 154
    - `Strategy` class, 229
  - `BeforeAppHooks` boot loader, 29
  - `before_app_loads` blocks, 29
  - `before_app_loads` plugin, 222
  - Belongs-to associations, 132–135

- be\_missing matcher, 291–292
- be\_successful matcher, 291–292
- BigDecimal class, 128
- Binary flags, storing as integers, 155
- Blank, testing for, 22
- blank? method, 22
- Block-aware enhancer, 103–105
- BlockAwareEnhancer, 103–105
- Blocks
  - calling later, 97
  - placing in worker queue, 74
- body method, 71
- body= method, 71
- Body of controller response, getting/setting, 71
- Boolean class, 128. *See also* TrueClass.
- Boolean property type, 128
- Boot loaders, overview, 27
- BootLoader class
  - AfterAppLoads, 30
  - BeforeAppHooks, 29
  - BuildFramework, 28–29
  - ChooseAdapter, 31
  - Cookies, 30
  - Defaults, 28
  - Dependencies, 29
  - DropPidFile, 28
  - LoadClasses, 29
  - Logger, 27
  - MimeTypes, 30
  - MixinSession, 29
  - RackUpApplication, 31–32
  - ReloadClasses, 32
  - ReloadTemplates, 32
  - SetupSession, 30
  - SetupStubClasses, 31
  - StartWorkerThread, 31
  - Templates, 29–30
- Bound variants *versus* unbound, 181
- bound\_check\_box method, 182–183

- Builder module, 179
- BuildFramework boot loader, 28–29
- build\_request method, 275–277
- build\_url method, 275–277

## C

- cache action, 274–275
- cache! action, 274–275
- cache\_action action, 274–275
- Caching
  - configuring
    - ActionStore, 270
    - AdhocStore, 270
    - FileStore, 269–270
    - fundamental stores, 269–270
    - GzipStore, 270
    - MemcachedStore, 269–270
    - PageStore, 270
    - SHA1Store, 270
    - strategy stores, 270
  - delete method, 273
  - deleting, 273
  - exists? method, 272
  - fetch methods, 273
  - fetching, 273
  - helpers
    - action caching, 274–275
    - build\_request method, 275–277
    - build\_url method, 275–277
    - cache action, 274–275
    - cache! action, 274–275
    - cache\_action action, 274–275
    - eager caching, 275–277
    - eager\_cache method, 275–277
    - fetch\_fragment method, 277
    - fetch\_partial method, 277–278
    - fragment caching, 277
    - partial caching, 277–278

- Caching (*continued*)
  - read method, 272
  - reading, 272
  - writable? method, 271–272
  - write method, 271–272
  - writing, 271–272
- Caitlin, Hampton, 105
- callable\_actions method, 72
- Callbacks, authentication, 236–237
- Camel case, converting strings to, 21
- camel\_case method, 21
- capture method, 71
- capture\_erb method, 71
- Cardinality, specifying, 136
- Casting values, 128–129
- catch\_content method, 71
- cattr\_accessor method, 20
- cattr\_reader method, 20
- cattr\_writer method, 20
- C flag, 9–10
- Checkbox control, 182–183
- check\_box method, 189–190
- Checking routes, 45
- check\_request\_for\_route method, 11
- ChooseAdapter boot loader, 31
- Class class, 20, 128
- Classes
  - listing, 20
  - models
    - creating, 116
    - inheritance, 116
    - organizing, 115
    - Principle of Substitutability, 116
    - required for DataMapper, 115–117
    - Resource method, 115–117
  - names
    - Haml views, 106–107
    - storing as strings, 128
  - reloading, 32
- class\_inheritable\_accessor method, 20
- class\_inheritable\_reader method, 20
- class\_inheritable\_writer method, 20
- class\_provided\_formats method, 72–73
- class\_provided\_formats= method, 72–73
- clear! method, 208
- clear\_content method, 71
- Clearing content, rendering templates, 71
- clear\_provides method, 72
- Client response time, filters, 82
- close\_sandbox! method, 11
- Collections, reversing object order, 145
- Comments, ERB views, 103
- Compiling templates, 89–90
- concat method, 71
- concat\_erb method, 71
- Conditions, validations, 163–164
- conditions query parameter, 145
- config/ directory, 12
- Config method, 14
- Configuration
  - caching
    - ActionStore, 270
    - AdhocStore, 270
    - FileStore, 269–270
    - fundamental stores, 269–270
    - GzipStore, 270
    - MemcachedStore, 269–270
    - PageStore, 270
    - SHA1Store, 270
    - strategy stores, 270
  - mailers
    - delivery methods, 254–255
    - quota checking, 255
    - sendmail, 254

- SMTP, 253–254
  - `test_send` method, 254
- sessions, 206
- Configuration files
  - databases, 19–20
  - `database.yml`, 19–20
  - `development.rb`, 16
  - environments, 16–17
  - init script
    - `after_app_loads` method, 13–14
    - basic configuration, 14
    - dependencies, 13–14
    - gems and load path, 12
    - inflector customization, 15
    - libraries, distributing, 12–13
    - ORM options, 14–15
    - plugins, distributing, 12–13
    - RubyGems, 12–13
    - template engines, 14
    - testing options, 15
  - `init.rb`, 19–20
  - logging
    - debug messages, 18
    - error messages, 18
    - example, 18
    - fatal messages, 18
    - flushing logs, 11, 18
    - info messages, 18
    - log file location, 17
    - message levels, specifying, 18
    - viewing logs, 18
    - warn messages, 18
  - `production.rb`, 16–17
  - router, 17
  - `router.rb`, 17, 43
  - routers, 43
  - `test.rb`, 17
  - YAML files, 19–20
- Console methods, 10–11
- Console traps, 9–10
- Constants, converting to paths, 21
- Content type for controller response,
  - getting/setting, 71
- `content_type` method, 71
- `content_type=` method, 71
- Contexts, validations, 164
- Controller access, sessions, 220
- Controller class
  - class methods
    - `callable_actions`, 72
    - `class_provided_formats`, 72
    - `class_provided_formats=`, 72
    - `clear_provides`, 72
    - `does_not_provide`, 72
    - `format`, 72
    - general, 72
    - `hide_action`, 72
    - `only_provides`, 72
    - overview, 70
    - `provides`, 72
    - `reset_provides`, 72
    - `show_action`, 72
  - instance methods
    - `class_provided_formats`, 73
    - `class_provided_formats=`, 73
    - `cookies`, 73
    - `delete_cookie`, 73
    - `does_not_provide`, 73
    - `escape_xml`, 74
    - escaping, 74
    - `format`, 73
    - general, 73
    - `h`, 74
    - `headers`, 73
    - `html_escape`, 74
    - `nginx_send_file`, 74
    - `only_provides`, 73
    - other, 74

Controller class (*continued*)instance methods (*continued*)

- params, 73
- provides, 73
- rack\_response, 73
- redirect, 73
- render\_chunked, 74
- render\_deferred, 74
- render\_then\_call, 74
- request, 73
- run\_later, 74
- send\_chunk, 74
- send\_data, 74
- send\_file, 74
- session, 73
- session, 73
- set\_cookie, 73
- status, 73
- status=, 73
- stream\_file, 74

## Controller classes

## AbstractController, methods

- before, 69
- absolute\_url, 71
- action\_name, 70
- action\_name=, 70
- add\_filter, 69
- after, 69
- body, 71
- body=, 71
- capture, 71
- capture\_erb, 71
- catch\_content, 71
- clear\_content, 71
- concat, 71
- concat\_erb, 71
- content\_type, 71
- content\_type=, 71
- controller\_name, 69–70
- default\_layout, 70
- filter, 69

- general, 69, 70
- inherited, 69
- instance, 70
- layout, 70
- normalize\_filters!, 69
- partial, 71
- relative\_url, 71
- render, 70
- render, 71
- render\_all, 71
- render\_html, 71
- render\_js, 71
- render\_json, 71
- render\_options, 70
- render\_text, 71
- render\_xml, 71
- render\_yaml, 71
- skip\_after, 69
- skip\_before, 69
- skip\_filter, 69
- subclasses\_list, 69
- template\_roots, 70, 71
- template\_roots=, 70, 71
- throw\_content, 71
- throw\_content?, 71
- URL, 71
- url, 71
- view, 71

## Application, 72

## Controller, class methods

- callable\_actions, 72
- class\_provided\_formats, 72
- class\_provided\_formats=, 72
- clear\_provides, 72
- does\_not\_provide, 72
- format, 72
- general, 72
- hide\_action, 72
- only\_provides, 72
- overview, 70
- provides, 72

- `reset_provides`, 72
- `show_action`, 72
- Controller, instance methods
  - `class_provided_formats`, 73
  - `class_provided_formats=`, 73
  - `cookies`, 73
  - `delete_cookie`, 73
  - `does_not_provide`, 73
  - `escape_xml`, 74
  - `escaping`, 74
  - `format`, 73
  - `general`, 73
  - `h`, 74
  - `headers`, 73
  - `html_escape`, 74
  - `nginx_send_file`, 74
  - `only_provides`, 73
  - `other`, 74
  - `params`, 73
  - `provides`, 73
  - `rack_response`, 73
  - `redirect`, 73
  - `render_chunked`, 74
  - `render_deferred`, 74
  - `render_then_call`, 74
  - `request`, 73
  - `run_later`, 74
  - `send_chunk`, 74
  - `send_data`, 74
  - `send_file`, 74
  - `session`, 73
  - `session`, 73
  - `set_cookie`, 73
  - `status`, 73
  - `status=`, 73
  - `stream_file`, 74
- Exceptions, 31, 74
- Controller classes, custom
  - actions
    - callable, setting, 79
    - hiding/showing, 79
    - readability, improving, 78
  - controller location, 75–76
  - naming controllers, 76
  - organizing methods
    - action readability, improving, 78
    - making methods available to subclasses, 77–78
    - overview, 76
    - sharing nonaction methods, 77
- `controller` key, 42
- Controller prefixes, 59
- Controller response
  - body, getting/setting, 71
  - content type, getting/setting, 71
  - formats provided, 72, 73
- `controller_for_slice` method, 201
- `controller_name` method, 69–70
- `Controller#part` method, 266–267
- Controllers
  - abstract, 69–70. *See also* **AbstractController**.
  - application, 72–74
  - authentication
    - `auth` more, 245
    - `auth` password slices, 247–249
  - calling actions, 66
  - definition, 34
  - dispatching requests, 66–67
  - exceptions, 74, 88
  - initializing, 66–67
  - location, 75–76
  - mailers, 257–258
  - Merb, 70–72. *See also* **Controller** class.
  - names
    - adding to `subclasses_list`, 69
    - creating, 76
    - returning, 69–70
  - parts, 263–266
  - rack response, 73
  - for slices, 200–201

- `Controller#send_mail` method, 258–259
- Convention over configuration, 5
- Converting
  - constants to paths, 21
  - numbers to currency values, 169–170
  - strings to camel case, 21
  - time to `EpochTime`, 155
- Cookies
  - deleting, 73
  - functionality, loading, 30
  - getting, 73
  - session domain, setting, 206
  - session storage, 211–214
  - session store, encrypting, 206
  - setting, 73
- `Cookies` boot loader, 30
- `cookies` method, 73
- `CookieSession` class, 211–214
- Core application layouts, 3
- `core` argument, 3
- `count` method, 159–161
- Counting numerical properties, 159–161
- `coverage` task, 282
- `create` action, 62
- `create_model_storage` method, 123–126
- `create_table_statement` method, 127
- CRUD (create, retrieve, update, delete) basics
  - `all` method, 142–145
  - `assert_kind` method, 144
  - creating records, 140–142
  - deleting collections, 154
  - deleting records, 154
  - `destroy!:` method, 154
  - `destroy` method, 154
  - enforcing types, 144
  - `first` method, 142–145
  - lazy loading of collections, 146–148
  - lazy loading of properties, 148–150
  - `LazyArray` class, 147–148
  - model objects, creating and saving, 140–142
  - N+1 queries, 150
  - overview, 140
  - query object algebra, 144
  - query parameters, adding, 143
  - `Query#merge` method, 144–145
  - `Relationship` class, 151–152
  - retrieving records
    - data sets, reloading, 146
    - fields, fetching as an array, 145
    - grouping by fields, 146
    - including other data, 145
    - lazy loading of collections, 146–148
    - limiting number returned, 146
    - links in related model data, 146
    - overview, 142–145
    - query offset, 146
    - query order, 146
    - reversing object order, 145
    - SQL conditions, setting, 145
    - strategic eager loading, 150–152
  - `scoped_query` method, 143
  - special query parameters, 145–146
  - strategic eager loading, 150–152
  - updating records
    - `attribute_set` method, 152
    - original values *versus* dirty values, 153
    - overview, 152
    - `save` method, 152
    - `update_attributes` method, 152–153
    - `update_speed` method, 153
- CSS error class, adding, 186
- `Csv` property type, 155
- Curly braces (`{ }`), in `HamL` tags, 107
- Currency values, converting numbers to, 169–170
- `current_form_context` method, 187–188

- Custom application layouts, 3
- Customizations, authentication, 236
- `customize_default` method, 236
- Cycle helpers, 176–177
- `cycle` method, 176–177
- Cycling through a list of values, 176–177

## D

- Data sets, reloading, 146
- Database storage
  - automigrating DB schema, 121–127
  - default configuration file, 120
  - overview, 120–121
  - `sample_development.db` file, 120
- Databases
  - accessing without a password, 19
  - configuration files, 19–20
  - creating with `rake`, 120
- `database.yml` files, 19–20, 115
- DataMapper
  - adapters, 114
  - associations, *versus* ActiveRecord, 132–133
  - CRUD. *See* CRUD (create, retrieve, update, delete) basics.
  - data sources, 114
  - database back ends, 114
  - definition, 26
  - Ferret adapter, 114
  - required module, 115–117
  - Salesforce adapter, 114
  - session storage, 217–219
- Date and time. *See also* Helpers, date and time.
  - converting time to EpochTime, 155
  - property type, 128
  - time equivalent of date, 22
- `Date` class, 128
- `date` format, 171
- `DateAndTimeFormatting` module, 173
- `DateTime` class, 22, 128
- `db` format, 171

- Debug messages, logging, 18
- Debugging code. *See* Interactive sessions.
- Decimal places, setting, 132
- Decimal precision, property type, 128
- `default` method, 57
- `:default` option, 131
- `default_cookie_domain` setting, 206
- `default_layout` method, 70
- Defaults boot loader, 28
- Deferred blocks, running, 31
- `deferred` method, 82
- Deferred routes, routing match rules, 53–54
- Deferring rendering responses, 74
- Defining property types, 127–130, 155–156
- Delete button, creating, 191
- `delete` method, 11, 273
- DELETE verb, 39–40
- `delete_button` method, 191
- `delete_cookie` method, 73
- Deleting
  - cache data, 273
  - collections, 154
  - cookies, 73
  - records, 154
  - URLs, 11
  - users, 62
- Dependencies
  - gems as, 13–14
  - init script file, 13–14
  - loading, 29
  - models, 114
- Dependencies boot loader, 29
- `dependencies` method, 14
- `dependency` method, 13
- Design principles, 5
- `destroy!`: method, 154
- `destroy` action, 62, 248
- `destroy` method, 154
- `destroy_model_storage` method, 123–126



Development environment, applications, 7,  
16–17

`development.rb` files, 16

Dirty values  
    *versus* original, 153  
    tracking properties for, 132

Discriminator class, 128

`_dispatch` method, 66

Dispatcher, overview, 33

Dispatching requests, 11, 66–67

`dispatch_request` method, 11

`dispatch_to` method, 11

`display` method, 95–96

`dm-aggregate` plugin, 158–161

`dm-timestamps` gem, 156–158

`does_not_provide` method, 72–73

Domain matching, 41

Dot (`.`), in class names, 106

`DropPidFile` boot loader, 28

`drop_table_statement` method, 126

Dual-formatted email messages, 255

## E

Eager caching, 275–277

`eager_cache` method, 275–277

Ebb adapter, 25

`ebb` key, 25

`edit` action, 62

`-e` flag, 16

Email. *See* Mailers.

`emongrel` key, 25

`encrypt` method, 243

Encryption  
    cookies session store, 206  
    `encrypt` method, 243  
    passwords, 243–244  
    strings, 155

Enforcing types, 144

`ensure_authenticated` helper,  
234–235

`Enum` property type, 155

Enumerated values, storing as integers, 155

Environment configuration file, loading, 29

Environments  
    applications, 7, 16–17  
    configuration files, 16–17  
    settings, 16–17  
    specifying, 16–17

`EpochTime`, converting to, 155

`EpochTime` property type, 155

Equal sign (`=`)  
    outputting results, 107–108  
    outputting strings, 108  
    sanitizing, 108

Equal signs (`==`), sanitizing, 108

ERB format, rendering templates, 89

ERB (eRuby) views. *See also* Views.  
    `#` (pound sign), comment indicator,  
    103  
    `<% %>` delimiters, 102  
    `<% =%>` delimiters, 102  
    `<%= %>` delimiters, 102  
    `<%= -%>` delimiters, 102  
    basic delimiters, 102  
    block-aware enhancer, 103–105  
    `BlockAwareEnhancer`, 103–105  
    comments, 103  
    overview, 101–102  
    whitespace, removing, 102

Error messages  
    logging, 18  
    storing, 232–233

`error_messages_for` method, 186, 191,  
250

`escape_regex` method, 21

`escape_xml` method, 74

Escaping  
    HTML entities, 74  
    methods, 74  
    special characters, 21, 155

- strings, 155
- XML entities, 74
- Evented Mongrel adapter, 25
- Exception controllers, setting up, 31
- Exceptions
  - in controller code, 74, 87–88. *See also* Exceptions controller.
  - overview, 87
  - raising, 87
- Exceptions class, 31, 74
- Exceptions controller, 247–249
- exists? method, 272
- Expiration time for sessions, setting, 206
- Extlib. *See also* Stack.
  - / (slash), in file paths, 21
  - / (slash), path expansion, 22
  - blank, testing for, 22
  - blank? method, 22
  - camel\_case method, 21
  - cattr\_accessor method, 20
  - cattr\_reader method, 20
  - cattr\_writer method, 20
  - class, 20
  - Class class, 20
  - classes, listing, 20
  - class\_inheritable\_accessor method, 20
  - class\_inheritable\_reader method, 20
  - class\_inheritable\_writer method, 20
  - date, 22
  - date, time equivalent, 22
  - DateTime class, 22
  - escape\_regex method, 21
  - Hook module, 23
  - in? method, 21
  - inheritable class-level variables, creating, 20
  - keys, strings or symbols as, 22
  - lazy loading, 23
  - LazyArray class, 23
  - Logger class, 22
  - logging, 22
  - Mash class, 22
  - meta\_class method, 21
  - methods, associating before/after other methods, 23
  - Object class, 21
  - objects
    - in arrays, duplicating, 21
    - in arrays, finding, 21
    - singleton class access, 21
  - ObjectSpace class, 20
  - overview, 20
  - Pathname class, 22
  - Pooling module, 23
  - resource sharing, 23
  - SimpleSet class, 23
  - snake\_case method, 21
  - String class, 21
  - strings
    - camel case, converting to, 21
    - escaping special characters, 21
    - as inline templates, 23
    - joining in file paths, 21
    - language translation, 21
    - path/constant conversion, 21
    - snake case, converting to, 21
  - String.translate method, 21
  - String.translations method, 21
  - t method, 21
  - time, returning, 22
  - Time class, 22
  - to\_const\_path method, 21
  - to\_const\_string method, 21
  - to\_datetime method, 22
  - to\_json method, 22
  - to\_time method, 22
  - try\_dup method, 21

Extlib. *See also* Stack. (*continued*)

- `unescape_regex` method, 21
- unified class and instance variables, creating, 20
- `VirtualFile` class, 23

## F

- `fake_request` method, 11
- `FalseClass` method, 22
- FastCGI adapter, 25
- Fatal messages, logging, 18
- `fcgi` key, 25
- Ferret adapter, 114
- `fetch` methods, 273
- `fetch_fragment` method, 277
- Fetching cache data, 273
- `fetch_partial` method, 277–278
- `-f` flag, 18
- `:field` option, 131
- Fields
  - fetching as an array, 145
  - grouping retrieved data by, 146
  - names, overriding, 131
- `fields` query parameter, 145
- Fieldset elements, creating, 180–182
- `fieldset` method, 180, 189
- `fieldset_for` method, 189
- `fields_for` method, 189
- File paths, strings in, 21
- `file_field` method, 189–190
- `FilePath` property type, 155
- Files, attaching to email, 260–261
- `FileStore` cache store, 269–270
- Filter chains, adding filters, 69
- Filter methods, 69
- Filters
  - before, 80–81
  - `AbstractController` methods, 69
  - adding to a filter chain, 69
  - after, 81–82
  - `after` class method, 81–82
  - after filter chain, 69
  - applying selectively, 82–83
  - `before` class method, 80–81
  - client response time, 82
  - `deferred` method, 82
  - ensuring option values are arrays, 69
  - before filter chain, 69
  - Haml views, 109
  - `:if` option, 83
  - options, 82–83
  - overview, 79–80
  - passing parameters to, 83
  - `skip_after` class method, 83–84
  - `skip_before` class method, 83–84
  - skipping, 69, 83–84
  - underscored, 248
  - `:unless` option, 83
  - `:with` option, 83
- `finalize` method
  - `SessionContainer` class, 208
  - `SessionStoreContainer` class, 211
- `first` method, 142–145
- Fixatable routes, 59–60
- `Flag` property type, 155
- Flat application layouts, 3, 6–8
- `flat` argument, 3
- Flexible segmentation, 52
- `Float` class, 128
- Floating property type, 128
- `flush` method, 18
- Flushing logs, 11, 18
- `form` method, 180, 188
- `Form` module, 187–191
- `format` key, 42
- `:format` key, 163
- Format methods, 72–73
- `:format` option, 132

## Formats

- controller response, 72, 73
- date and time, 171–173
- rendering templates
  - default, 89
  - ERB, 89
  - specifying, 92–94

`form_contexts` method,  
187–188

`form_for` method, 188–189

## Forms

- actions, setting automatically, 187
- context, changing from another
  - form, 189
- default builder, 187
- elements, creating, 180–182, 185–187.  
*See also* Helpers, form.
- helpers. *See* Helpers, form.
- names, 188–189
- without model object content, 188

`ForumSlice` module, 196

Fragment caching, 277

Freezing gems, 13

## G

Garbage collection, session storage, 215–216

## Gems

- as dependencies, 13–14
- directory size, reducing, 13
- freezing, 13
- libraries, distributing, 12–13
- and load path, init script file, 12
- `--no-rdoc` flag, 13
- plugins, distributing, 12–13
- RubyGems, 12–13
- version-specific, installing, 13

`generate` method

- `SessionContainer` class, 208
- `SessionStoreContainer` class, 211

`get` method, 11

GET verb, 39–40

Getter/setter methods, dynamic creation, 119

Getting. *See* Retrieving; *specific items*.

Globs, 29

`GzipStore` strategy store, 270

## H

`h` method, 74

`halt!` method, 230

`halted?` method, 230

`:haml` parameter, 14

Haml views. *See also* Views.

- `=` (equal sign)
  - outputting results, 107–108
  - outputting strings, 108
  - sanitizing, 108
- `.` (dot), in class names, 106
- `==` (equal signs), sanitizing, 108
- `-` (hyphen), code indicator, 107
- `%` (percent sign), tag indicator, 105
- `#` (pound sign), in ID names, 106
- `{ }` (curly braces), in Haml tags, 107
- `~` (tilde), preserving whitespace, 108
- class names, 106–107
- filters, 109
- history of, 105
- HTML injections, preventing, 108
- HTML-sensitive characters, sanitizing,  
108
- ID names, 106–107
- indentation, 106
- versus* inline styles, 109
- interpreting lines, 107
- nesting code, 106
- outputting lines, 107–108
- sanitized lines, 108
- tags, 105–106
- whitespace, preserving, 108

Has associations, 132–133, 135–137

**has** method, 132–133

Has through associations, 138–139

**have\_body** matcher, 292

**have\_content\_type** matcher, 292

Headers, requests, 73

**headers** method, 73

Helpers

authentication, 234–235

caching

action caching, 274–275

**build\_request** method, 275–277

**build\_url** method, 275–277

**cache** action, 274–275

**cache!** action, 274–275

**cache\_action** action, 274–275

eager caching, 275–277

**eager\_cache** method, 275–277

**fetch\_fragment** method, 277

**fetch\_partial** method, 277–278

fragment caching, 277

partial caching, 277–278

cycle, 176–177

**cycle** method, 176–177

date and time

date format, 171

**DateAndTimeFormatting** module,  
173

db format, 171

formats, 171–173

long format, 171

**prettier\_time** method, 175–176

relative date and time, 175–176

relative time, 175–176

**relative\_date** method, 175

**relative\_date\_span** method, 175

**relative\_time\_span** method, 175

RFC 822 format, matching, 171

**rfc822** format, 171

short format, 171

Time DSL, 173–175

time format, 171

**time\_lost\_in\_words** method, 175

numeric

**minutes\_to\_hours** method, 169

**to\_currency** method, 169–170

**two-digits** method, 168

**ordinalize** method, 173

ordinals, 173

overview, 167

tag, 177–178

**tag** method, 178

**truncate** method, 167–168

Helpers, form. *See also* Forms.

builders

bound variants *versus* unbound, 181

**bound\_check\_box** method,  
182–183

**Builder** module, 179

checkbox control, 182–183

CSS error class, adding, 186

**error\_messages\_for** method, 186

fieldset elements, 180–182

**fieldset** method, 180

form actions, setting automatically, 187

form elements, 180–182, 185–187

**form** method, 180

forms, default builder, 187

label elements, 184–185

**label** method, 184–185

name parameter, 179

obj parameter, 179

origin parameter, 179

**process\_form\_attrs** method,  
180, 187

**ResourcefulFormWithErrors**, 187

**update\_bound\_check\_box** method,  
183–184

- `update_bound_controls`
      - method, 186
    - `update_*_controls` method,
      - 181–182
    - validation errors, outputting, 186
  - helpers
    - `check_box` method, 189–190
    - `current_form_context` method,
      - 187–188
    - delete button, creating, 191
    - `delete_button` method, 191
    - `error_messages_for` method, 191
    - `fieldset` method, 189
    - `fieldset_for` method, 189
    - `fields_for` method, 189
    - `file_field` method, 189–190
    - form context, changing from another
      - form, 189
    - `form` method, 188
    - Form module, 187–191
    - form names, 188–189
    - `form_contexts` method, 187–188
    - `form_for` method, 188–189
    - forms without model object content, 188
    - `hidden_field` method, 189–190
    - `password_field` method, 189–190
    - `radio_button` method, 189–190
    - `radio_group` method, 189–190
    - `select` method, 189–190
    - singleton form context, 188
    - `text_area` method, 189–190
    - `text_field` method, 189–190
    - validation errors, outputting, 191
  - overview, 179
  - `hidden_field` method, 189–190
  - `hide_action` method, 72
  - Hiding actions, 72
  - Hijacking user accounts, 59–60
  - Hook module, 23
  - Hooking methods, authentication, 224
  - Hooks, models, 154
  - HTML (HyperText Markup Language)
    - email messages, 255
    - entities, escaping, 74
    - injections, preventing, 108
    - MIME set, rendering views with, 71
  - html task, 282
  - `html_escape` method, 74
  - HTML-sensitive characters, sanitizing, 108
  - HTTP verbs, 39–40
  - Hyphen (-), code indicator, 107
- I**
- ID names, Haml views, 106–107
  - `identify` method, 63
  - `:if` option, 83
  - `-i` flag, 9–10, 13
  - `in?` method, 21
  - `includes` query parameter, 145
  - Indentation, Haml views, 106
  - `index` action, 61
  - `:index` option, 132
  - Indexing properties, 132
  - Inflector customization, 15
  - info messages, logging, 18
  - Inheritable class-level variables, creating, 20
  - Inheritance, model classes, 116
  - `inherited` method
    - `AbstractController` class, 69
    - `Strategy` class, 229
  - Init file, loading, 29
  - Init script file
    - `after_app_loads` method, 13–14
    - basic configuration, 14
    - dependencies, 13–14
    - gems and load path, 12
    - inflector customization, 15
    - libraries, distributing, 12–13

- Init script file (*continued*)
  - ORM options, 14–15
  - plugins, distributing, 12–13
  - RubyGems, 12–13
  - template engines, 14
  - testing options, 15
- initialize** method, 230
- Initializing controllers, 66–67
- init.rb** files, 19–20
- Inline styles *versus* Haml views, 109
- Inlining templates, 90
- Instance methods, 70
- Integer** class, 128
- Integers. *See also* Numbers.
  - auto-incrementing, 155
  - binary flags as, 155
  - enumerated values as, 155
  - property type, 128
- Interactive Merb, registering routes, 45–48
- Interactive sessions
  - check\_request\_for\_route**
    - method, 11
  - close\_sandbox!** method, 11
  - console methods, 10–11
  - console traps, 9–10
  - delete** method, 11
  - dispatch\_request** method, 11
  - dispatch\_to** method, 11
  - fake\_request** method, 11
  - get** method, 11
  - logger output, flushing, 11, 18
  - open\_sandbox!** method, 11
  - post** method, 11
  - put** method, 11
  - reload!** method, 11
  - reloading the application, 11
  - request** method, 11
  - request routes, checking or listing, 11
  - requests, 11

- sandboxing, 11–12
- show\_routes** method, 11
- starting, 9
- trace\_log!** method, 11
- url** method, 11
- URLs, 11
- IP addresses, storing as strings, 155
- IPAddress** property type, 155
- IRB adapter, 25
- irb** key, 25
- Iterating through a list of values, 176–177

## J

- Join table models, creating, 139
- JS MIME set, rendering views with, 71
- JSON, storing as a string, 155
- JSON MIME set, rendering views with, 71
- Json** property type, 155

## K

- Katz, Yehuda, 103, 114
- Kernel** methods, 293–294
- :key** option, 131
- Keys
  - action**, 42
  - controller**, 42
  - ebb**, 25
  - emongrel**, 25
  - fcgi**, 25
  - format**, 42
  - :format**, 163
  - irb**, 25
  - :key** option, 131
  - :length**, 163
  - :nullable**, 163
  - Rack, 25
  - route parameters, 42
  - runner**, 25

- `session_id_key` setting, 206
- `session_secret_key` setting, 206
- `:set`, 163
- `:size`, 163
- strings or symbols as, 22
- `swift`, 25
- table keys, defining, 131
- `thin`, 25
- `webrick`, 25

## L

- Label elements, creating, 184–185
- `label` method, 184–185
- Language translation, 21
- `layout` method, 70
- Layout options, rendering templates
  - class-level, setting, 70
  - disabling, 70
  - resetting, 70
  - setting, 70
- Layout templates, 94–95
- `layout_for_slice` method, 201
- Lazy loading
  - collections, 146–148
  - enabling, 131
  - `:lazy` option, 131
  - `LazyArray` class, 23, 147–148
  - properties, 148–150
  - `:lazy` option, 131
- `LazyArray` class, 23, 147–148
- `:length` key, 163
- `:length` option, 131. *See also* `:size` option.
- `lib` files, 246–247
- Libraries, distributing, 12–13
- `limit` query parameter, 146
- `links` query parameter, 146
- Listing
  - classes, 20
  - routes

- with audit routes rake task, 48
- `audit:routes` command, 48
- with interactive Merb, 45–48
- `route` command, 45
- `show_routes` method, 11, 47–48
- subclasses, 69
- users, 61
- Literal matching, routing match rules, 50–51
- Literal segments, 40–41
- `LoadClasses` boot loader, 29
- Location, controllers, 75–76
- `Logger` boot loader, 27
- `Logger` class, 22
- Logger output, flushing, 11, 18
- Loggers, updating, 29
- Logging
  - configuration files
    - debug messages, 18
    - error messages, 18
    - example, 18
    - fatal messages, 18
    - flushing logs, 11, 18
    - info messages, 18
    - log file location, 17
    - message levels, specifying, 18
    - viewing logs, 18
    - warn messages, 18
  - Extlib, 22
  - `Logger` boot loader, 22, 27
- Login view, 249–250
- `login_param` method, 239
- Logins, redirecting, 245
- long format, 171

## M

- `MailController` classes, 257–258
- `MailController#attach` method, 260–261



- Mailer** class
  - configuring, 253–255
  - MailFactory interface, 255–257
- Mailers**
  - actions
    - invoking, 258–259
    - naming, 257–258
  - attaching files, 260–261
  - configuration
    - delivery methods, 254–255
    - quota checking, 255
    - sendmail, 254
    - SMTP, 253–254
    - `test_send` method, 254
  - controllers, 257–258
  - `Controller#send_mail` method, 258–259
  - dual-formatted messages, 255
  - generating mailer files, 261–262
  - HTML email, 255
  - MailController** classes, 257–258
  - `MailController#attach` method, 260–261
  - MailFactory, 255–257
  - multipart messages, 255
  - parameters, 259–260
  - rendering message views, 261
  - templates, 261
  - testing, 261
    - `:test_method` method, 261
  - using directly, 255–257
- MailFactory, 255–257
- ManyToMany** associations, 139
- Many-to-many-through associations, 139
- Mash** class, 22
- Master process, 2
- Match captures, registering routes with, 56
- `match` method, routing match rules, 50
- Match rules. *See* Routing, match rules.
- `max` method, 159
- Maximum value, calculating, 159
- Memcached session storage, 216–217
- MemcachedStore** cache store, 269–270
- Memory session storage, 214–216
- MemorySessionStore** class, 215–216
- Merb controllers, 70–72
- Merb servers
  - overview, 32–33
  - shutting down, 2
- Merb stack. *See* Stack.
- merb-gen** command, 1–2
- `meta_class` method, 21
- (Method) parameters, 43
- Method route conditions, 39–40
- Methods. *See also* Helpers; *specific methods*.
  - associating before/after other methods, 23
  - available as actions, 68
  - organizing
    - action readability, improving, 78
    - making methods available to subclasses, 77–78
    - overview, 76
    - sharing nonaction methods, 77
- MIME types, registering defaults, 30
- MimeTypes** boot loader, 30
- `min` method, 159
- Minimum value, calculating, 159
- `minutes_to_hours` method, 169
- MixinSession** boot loader, 29
- Model class, selecting for authentication, 223
- Model objects, creating/saving, 140–142
- Model serial IDs, property type, 128
- Model specs, in testing, 283–285
- model** task, 281
- Models
  - `after` method, 154
  - associations. *See* Associations.
  - authentication, `auth more`, 242–245

- classes
  - creating, 116
  - inheritance, 116
  - organizing, 115
  - Principle of Substitutability, 116
  - required for `Mapper`, 115–117
  - `Resource` method, 115–117
- configuration, 113–115
- CRUD basics. *See* CRUD (create, retrieve, update, delete) basics.
- `database.yml` file, 115
- `Mapper`
  - adapters, 114
  - data sources, 114
  - database back ends, 114
- dependencies, 114
- hooks, 154
- `before` method, 154
- plugins. *See* Plugins.
- properties. *See* Properties.
- repository configuration, 114–115
- serial IDs, 128
- YAML configuration file, 114–115
- YAML nodes, 115
- Mongrel adapter, 25
- Multipart email messages, 255

## N

- N+1 queries, 150
- `name` method, 58
- `name` parameter, 179
- Names
  - actions, 70
  - controllers
    - adding to `subclasses_list`, 69
    - creating, 76
    - returning, 69–70
  - mail controller actions, 257–258
  - `MailController` classes, 257

- routes
  - adding, 57–58
  - controller prefixes, 59
  - namespaces, 59
  - prefixing, 58–59
- `namespace` method, 59
- Namespaces, 59
- Neighman, Daniel, 221
- Nesting match statements, 50
- `new` action, 62
- `nginx_send_file` method, 74
- `NilClass` class, 22
- Nontemplate code, loading, 29
- `--no-rdoc` flag, 13
- `normalize_filters!` method, 69
- Null values, disallowing, 131
- `:nullable` key, 163
- `:nullable` option, 131
- Numbers. *See also* Integers.
  - average value, calculating, 159
  - Boolean, 128
  - converting to currency, 169–170
  - counting, 159–161
  - dates and times, 128
  - decimal, precision, 128
  - decimal places, setting, 132
  - floating, 128
  - integers, 128
  - maximum value, calculating, 159
  - minimum value, calculating, 159
  - model serial IDs, 128
  - ordinals, 173
  - property types, 128
  - single-digit, padding to two digits, 168
  - summing, 159
- `Numeric` class, 22
- Numeric helpers
  - `minutes_to_hours` method, 169
  - `to_currency` method, 169–170
  - `two-digits` method, 168

## O

- `obj` parameter, 179
- `Object` class
  - `blank?` method, 22
  - in Extlib, 21
  - property type, 128
- Objects
  - in arrays, duplicating, 21
  - in arrays, finding, 21
  - marshaling into records, 128
  - singleton class access, 21
- `ObjectSpace` class, 20
- `offset` query parameter, 146
- One-to-many-through associations, 138–139
- `only_provides` method, 72–73
- OpenID, authentication strategy, 242
- `open_sandbox!` method, 11
- Optional matching, routing match rules, 52–53
- `options` method, 59
- `order` query parameter, 146
- `ordinalize` method, 173
- Ordinals, 173
- `origin` parameter, 179
- ORM options, init script file, 14–15
- ORMs (object relational mappings), 25–27

## P

- `PageStore` strategy store, 270
- Parameters
  - mailers, 259–260
  - (Method) parameters, 43
  - passing to filters, 83
  - request parameters, 43
  - from requests, 73
  - route parameters, 43
- `params` method, 73
- Parentheses (`()`), in optional route matching, 52
- `PartController` class, 263–266
- Partial caching, 277–278

- `partial` method, 71
- Partials, 111–112
- Parts
  - actions, 266–267
  - controllers, 263–266
  - generating files, 267
- Passing in literal strings, routing match rules, 51
- Password form, authentication strategy, 241
- `password_field` method, 189–190
- `password_param` method, 239
- Passwords. *See also* Authentication, `auth`
  - password slices; Security.
    - accessing databases without, 19
    - authentication. *See* Authentication, `auth`
      - password slices.
    - encrypting, 243–244
    - encryption, 243–244
    - form authentication strategy, 241
    - `password_field` method, 189–190
    - `password_param` method, 239
- Path route conditions
  - definition, 39
  - literal segments, 40–41
  - overview, 40–41
  - query strings, 40
  - segments, 40–41
  - symbolic segments, 40–41
  - syntax, 40–41
- `Pathname` class, 22
- Paths
  - converting to constants, 21
  - storing as strings, 155
- Percent sign (%), tag indicator, 105
- Persistent accessors, 118
- Pidfile, dropping, 28
- Plugins
  - addressable URIs, storing as strings, 155
  - aggregates, 158–161
  - auto-incrementing integers, 155

- average value, calculating, 159
- `avg` method, 159
- binary flags, storing as integers, 155
- `count` method, 159–161
- counting numerical properties, 159–161
- distributing, 12–13
- `dm-aggregate` plugin, 158–161
- `dm-timestamps` gem, 156–158
- encrypting strings, 155
- enumerated values, storing as integers, 155
- EpochTime, converting to, 155
- escaping strings, 155
- IP addresses, storing as strings, 155
- JSON, storing as a string, 155
- `max` method, 159
- maximum value, calculating, 159
- `min` method, 159
- minimum value, calculating, 159
- numerical properties, 159
- parsing strings as CSVs, 155
- paths, storing as strings, 155
- property types, 155–156. *See also specific types.*
- records, counting, 158–161
- regular expressions, storing as strings, 155
- strings, 155
- `sum` method, 159
- summing numerical values, 159
- timestamps, 156–158
- URLs, strings in, 155
- UUIDs, storing as strings, 155
- validations. *See also specific validation methods.*
  - automatic, turning off, 132, 163
  - with blocks, 162
  - conditions, 163–164
  - confirming attributes, 162
  - contexts, 164
  - errors, 164–165
  - format of a value, 162
  - formats, 163
  - with methods, 162
  - numerical length, 162–163
  - numerical values, 162
  - overview, 162–163
  - presence of, 163
  - presence of a value, 162
  - within a range of values, 162–163
  - true values, 162
  - uniqueness, 162–163
  - `validates_within` method, 163
- YAML, storing as a string, 155
- Pooling module, 23
- `post` method, 11
- POST verb, 39–40
- Posting URLs, 11
- Pound sign (#)
  - comment indicator, 103
  - in ID names, 106
- Power, David, 255
- `:precision` option, 132
- Prefixing route names, 58–59
- Prepare block, router configuration, 43–44
- `prettier_time` method, 175–176
- Principle of least surprise, 5
- Principle of Substitutability, 116
- `process_form_attrs` method,
  - 180, 187
- Production environment, applications, 7,
  - 16–17
- `production.rb` files, 16–17
- Properties
  - access privileges, setting, 131
  - `alter_table_column_statement` method, 126
  - auto-increment, enabling, 131
  - `auto_migrate!` method, 123–126
  - `auto_upgrade!` method, 123–126
  - `create_model_storage` method,
    - 123–126

Properties (*continued*)

- `create_table_statement` method, 127
- database storage
  - automigrating DB schema, 121–127
  - default configuration file, 120
  - overview, 120–121
  - `sample_development.db` file, 120
- in DataMapper, benefits of, 121
- decimal places, setting, 132
- default value, setting, 131
- defining
  - casting values, 128–129
  - options, 131–132. *See also specific options.*
  - overview, 127
  - property types, 127–130, 155–156. *See also specific types.*
- `destroy_model_storage` method, 123–126
- `drop_table_statement` method, 126
- field names, overriding, 131
- getter/setter methods, dynamic creation, 119
- including in model files, 121
- indexing, 132
- lazy-loading, enabling, 131
- null values, disallowing, 131
- overview, 118–120
- persistent accessors, 118
- `properties_with_subclasses` method, 127
- `property_schema_statement` method, 127
- size, setting, 131
- with subclasses, 127
- table keys, defining, 131
- tracking for dirtiness, 132
- type plugins, 155–156
- `upgrade_model_storage` method, 123–126

- validation, 132
- values, setting with instance variables, 119–120
- `properties_with_subclasses` method, 127
- `property_schema_statement` method, 127
- Protocol matching, 41
- `provides` method, 72–73
- `public/` directory, 9
- `push_path` method, 222
- `put` method, 11
- PUT verb, 39–40
- Putting URLs, 11

## Q

## Queries

- N+1, 150
- parameters, adding, 143
- special query parameters, 145–146
- strings, path route conditions, 40
- Query object algebra, 144
- Query offset, 146
- Query order, 146
- `Query#merge` method, 144–145
- Quota checking, email, 255

## R

Rack. *See also* Stack.

- adapter options, 23–25
- adapters, choosing, 31
- Ebb adapter, 25
- ebb key, 25
- emongrel key, 25
- Evented Mongrel adapter, 25
- FastCGI adapter, 25
- fcgi key, 25
- functional description, 24–26
- IRB adapter, 25

- `irb` key, 25
- Mongrel adapter, 25
- ORMs (object relational mappings), 25
- overview, 23
- Runner adapter, 25
- `runner` key, 25
- setting up, 31–32
- `swift` key, 25
- Swiftpplied Mongrel adapter, 25
- Thin adapter, 25
- `thin` key, 25
- WEBrick adapter, 25
- `webrick` key, 25
- Rack response, controllers, 73
- `rack_response` method, 73
- `RackUpApplication` boot loader, 31–32
- `radio_button` method, 189–190
- `radio_group` method, 189–190
- Rake tasks, in testing, 279–282
- `read` method, 272
- `:reader` option, 131
- Reading cache data, 272
- Reaping session storage, 215–216
- Records, counting, 158–161
- `redirect` matcher, 292
- `redirect` method
  - `Controller` class, 73
  - registering a redirect, 56
  - `Strategy` class, 230
- `redirect_back_or` method, 245
- Redirecting logins, 245
- Redirects
  - after-POST requests, 85
  - a caveat, 84–85
  - in before filters, 86–87
  - overview, 84
  - `redirect` method, 73, 84
  - registering, 56
  - registering routes with, 56
- `redirect_to` matcher, 292
- `regenerate` method
  - `SessionContainer` class, 208
  - `SessionStoreContainer` class, 211
- `Regexp` property type, 155
- `register` method
  - generating slices, 195–196
  - registering routes with, 55–56. *See also* `To` method; `With` method.
- Registering route parameters, 54
- Registering routes with
  - match captures, 56
  - `to` method, 54–55. *See also* `Register` method; `With` method.
  - `with` method, 55. *See also* `Register` method; `To` method.
  - redirecting, 56
  - `register` method, 55–56. *See also* `To` method; `With` method.
  - symbols, 56–57
- Regular expressions
  - routing match rules, 53
  - segment-specific, routing match rules, 52
  - setting route conditions, 39
  - storing as strings, 155
  - validation against, 132
- `Relationship` class, 151–152
- Relative time, 175–176
- `relative_date` method, 175
- `relative_date_span` method, 175
- `relative_time_span` method, 175
- `relative_url` method, 71
- `reload!` method, 11
- `reload` query parameter, 146
- `ReloadClasses` boot loader, 32
- Reloading
  - applications, 11
  - classes, 32
- `ReloadTemplates` boot loader, 32
- Renaming resources, 63

- render method
  - `AbstractController` class, 71
  - basic rendering, 91–92
- Render methods, 70
- `render_all` method, 71
- `render_chunked` method, 74, 96
- `render_deferred` method, 74, 96–97
- `render_html` method, 71
- Rendering
  - email message views, 261
  - responses, 74
  - views, 71
- Rendering, templates
  - basic rendering, 91–92
  - clearing content, 71
  - compiling templates, 89–90
  - `display` method, 95–96
  - formats
    - default, 89
    - ERB, 89
    - specifying, 92–94
  - inlining templates, 90
  - layout options, 70
  - layout template, 94–95
  - overview, 89
  - `render` method, 91–92
  - render methods, 70
  - `render_chunked` method, 96
  - `render_deferred` method, 96–97
  - `render_then_call` method, 97
  - status code, setting, 94
  - `:status` option, 94
- `render_js` method, 71
- `render_json` method, 71
- `render_options` method, 70
- `render_text` method, 71
- `render_then_call` method, 74, 97
- `render_xml` method, 71
- `render_yaml` method, 71
- Repository configuration, 114–115
- Request access, sessions, 219
- `Request` class, 33
- Request helper, in testing, 289–290
- Request matchers, in testing, 290–292
- `request` method, 11, 73
- Request parameters, 43
- Request routes, checking or listing, 11
- Request specs, in testing, 285–288
- `request` task, 281
- `REQUEST_METHOD` option, 66–67
- `REQUEST_PATH` option, 66–67
- Requests
  - building, 11
  - dispatching, 11, 33, 66–67
  - headers, 73
  - incoming, dispatching, 33
  - object routed to the controller, 73
  - overview, 33
  - parameters, 73
  - returning, 11
  - status codes, 73
- `reset_provides` method, 72
- `Resource` method, 115–117
- `resource` method, 62–63
- Resource routes. *See also* Routers; Routes; Routing.
  - renaming resources, 63
  - standard resources routing
    - `create` action, 62
    - deleting users, 62
    - `destroy` action, 62
    - `edit` action, 62
    - `index` action, 61
    - `new` action, 62
    - `resources` method, 60–62
    - `show` action, 62
  - single resource routing, 62–63
  - `update` action, 62
  - updating users, 62

- user accounts, creating, 62
  - user profiles, displaying, 62
  - users, listing, 61
  - Resource sharing, 23
  - ResourcefulFormWithErrors**, 187
  - resources** method, 60–62
  - respond\_successfully** matcher, 291–292
  - Responses, authentication, 233–234
  - responses.rb** file, 233–234
  - Restarting after code changes, avoiding, 7
  - retrieve** method, 211
  - Retrieving
    - records
      - data sets, reloading, 146
      - fields, fetching as an array, 145
      - grouping by fields, 146
      - including other data, 145
      - lazy loading of collections, 146–148
      - limiting number returned, 146
      - links in related model data, 146
      - overview, 142–145
      - query offset, 146
      - query order, 146
      - reversing object order, 145
      - SQL conditions, setting, 145
      - strategic eager loading, 150–152
    - URLs, 11
  - RFC 822 format, matching, 171
  - rfc822** format, 171
  - route** command, 45
  - Route conditions
    - DELETE**, 39–40
    - domain matching, 41
    - GET**, 39–40
    - HTTP verbs, 40
    - methods, 39–40
    - minimal requirements, 39
    - paths
      - definition, 39
      - literal segments, 40–41
      - overview, 40–41
      - query strings, 40
      - segments, 40–41
      - symbolic segments, 40–41
      - syntax, 40–41
  - POST**, 39–40
  - protocol matching, 41
  - PUT**, 39–40
  - setting with regular expressions, 39
- Route parameters
  - action** key, 42
  - controller** key, 42
  - defaults, setting, 57
  - definition, 42
  - format** key, 42
  - keys, 42
  - registering, 54
  - settings, 54
- Router helper, authentication, 235
- router.rb** files, 17
- Routers
  - behaviors, definition, 38
  - configuration
    - configuration file, 43
    - prepare block, 43–44
    - route order, 44
    - router.rb** file, 43
    - routes, adding, 44–45
  - configuration files, 17
  - definition, 33
  - testing URL generation and recognition, 49–50
- Routes
  - absolute URLs, returning, 71
  - adding, 44–45
  - checking, 45
  - definition, 38
  - fixatable, 59–60
  - including session IDs. *See* Session fixation.



Routes (*continued*)

interaction with resources. *See* Resource routes.

## listing

with audit routes rake task, 48  
**audit:routes** command, 48  
 with interactive Merb, 45–48  
**route** command, 45  
**show\_routes** method, 47–48

## names

adding, 57–58  
 controller prefixes, 59  
 namespaces, 59  
 prefixing, 58–59

order, specifying, 44

## registering with

match captures, 56  
**to** method, 54–55. *See also* **Register** method; **With** method.  
**with** method, 55. *See also* **Register** method; **To** method.  
 redirecting, 56  
**register** method, 55–56. *See also* **To** method; **With** method.

symbols, 56–57

relative URLs, returning, 71

session fixation, 59–60

showing, 11, 47–48

## Routing, match rules

deferred routes, 53–54  
 full regular expressions, 53  
 literal matching, 50–51  
**match** method, 50  
 nesting match statements, 50  
 overview, 50  
 passing in literal strings, 51  
 symbolic matching

( ) (parentheses), in optional matching, 52  
 automatic parameters, 51

flexible segmentation, 52

optional matching, 52–53

overview, 51

segment-specific regular expressions, 52

Routing, overview, 38

RSpec extensions, in testing, 292–294

RubyGems, 12–13

**run!** method, 239

**run\_after\_authentication\_**  
     **callbacks** method, 226–227,  
     237

**run\_later** method, 97

Runner adapter, 25

**runner** key, 25

## S

Salesforce adapter, 114

**sample\_development.db** file, 120

Sandboxing, 11–12

Sanitized lines, Haml views, 108

**save** method, 152

**:scale** option, 132

Schaefer, Bernerd, 114

**scoped\_query** method, 143

Security. *See also* Passwords.

hijacking user accounts, 59–60

session fixation, 59–60

Segments, path route conditions, 40–41

Segment-specific regular expressions, routing  
     match rules, 52

**select** method, 189–190

**send\_chunk** method, 74

**send\_data** method, 74

**send\_file** method, 74

## Sending

binary data, 74

chunks, 74

files

attached to email, 260–261

overview, 97–98

- `send_file` method, 98
  - through `nginx`, 74
- Sendmail, 254
- Sequel, 26
- Serial callbacks, authentication, 226–227
- `Serial` class, 128
- Serial IDs, models, 128
- `:serial` option, 131
- Serial property type, 155
- Server. *See* Merb servers.
- Session containers, 207–208, 214–215
- Session fixation, 59–60
- `session` method, 73, 220
- Session methods, 73
- Session store, accessing, 73
- `SessionContainer` class, 207–208
- `session_expiry` setting, 206
- `session_id=` method, 208
- `session_id_key` setting, 206
- Sessions
  - authentication, 231
  - configuration, 206
  - containers, setting up, 30
  - controller access, 220
  - cookie domain, setting, 206
  - cookie session store, encrypting, 206
  - default values, setting up, 30
  - `default_cookie_domain` setting, 206
  - definition, 34
  - expiration time, setting, 206
  - functionality, adding, 29
  - IDs, 205–206
  - overview, 205–206
  - request access, 219
  - `session` method, 220
  - `session_expiry` setting, 206
  - `session_id_key` setting, 206
  - `session_secret_key` setting, 206
  - `session_store` setting, 206
- Sessions, storing
  - `clear!` method, 208
  - `finalize` method
    - `SessionContainer` class, 208
    - `SessionStoreContainer` class, 211
  - `generate` method
    - `SessionContainer` class, 208
    - `SessionStoreContainer` class, 211
  - `regenerate` method
    - `SessionContainer` class, 208
    - `SessionStoreContainer` class, 211
  - `retrieve` method, 211
  - session containers, 207–208, 214–215
  - `SessionContainer` class, 207–208
  - `session_id=` method, 208
  - `SessionStoreContainer` class, 208–211
  - `setup` method
    - `SessionContainer` class, 208
    - `SessionStoreContainer` class, 211
  - storage mechanisms
    - cookie sessions, 211–214
    - `CookieSession` class, 211–214
    - DataMapper sessions, 217–219
    - garbage collection, 215–216
    - memcached sessions, 216–217
    - memory sessions, 214–216
    - `MemorySessionStore` class, 215–216
    - reaping sessions, 215–216
    - `TamperedWithCookie` error, 214
  - store containers, 208–211, 214–215
- Sessions controller, 247–249
- `session_secret_key` setting, 206
- `session_store` setting, 206
- `SessionStoreContainer` class, 208–211
- `:set` key, 163
- `set_cookie` method, 73
- Setting. *See specific items.*

- setup method
  - SessionContainer class, 208
  - SessionStoreContainer class, 211
- setup.rb file, 222
- SetupSession boot loader, 30
- SetupStubClasses boot loader, 31
- SHA1Store strategy store, 270
- short format, 171
- show action, 62
- show\_action method, 72
- Showing
  - actions, 72
  - routes, 11, 47–48
- show\_routes method, 11, 47–48
- SimpleSet class, 23
- Single resource routing, 62–63
- Singleton form context, 188
- Singular words, switching to plural, 15
- :size key, 163
- Size of properties, setting, 131
- :size option, 131. *See also* :length option.
- skip\_after class method, 83–84
- skip\_after method, 69
- skip\_before class method, 83–84
- skip\_before method, 69
- skip\_filter method, 69
- Skipping filters, 69, 83–84
- Slash (/)
  - in file paths, 21
  - path expansion, 22
- slice command, 196–199
- slice method, 201–203
- Slices. *See also* Authentication, auth password slices.
  - developing
    - building slices into gems, 199–200
    - controller\_for\_slice method, 201
    - controllers, 200–201
    - ForumSlice module, 196
    - generating slices, 193–196
    - layout\_for\_slice method, 201
    - register method, 195–196
    - running slices, 196–199
    - slice command, 196–199
    - using
      - add\_slice method, 202–203
      - overview, 201
      - slice method, 201–203
- Slug property type, 155
- SMTP, 253–254
- Snake case, converting strings to, 21
- snake\_case method, 21
- Spec files, in testing, 282–283
- Special characters, escaping, 21
- spectasks.rb file, 279–280
- SQL conditions, setting, 145
- Stack. *See also* Extlib; Rack.
  - ORMs (object relational mappings), 26–27.
    - See also* ActiveRecord; Sequel.
  - plugins, 26–27
- Standard application layouts, 3, 8–9
- Standard resources routing
  - create action, 62
  - deleting users, 62
  - destroy action, 62
  - edit action, 62
  - index action, 61
  - new action, 62
  - resources method, 60–62
  - show action, 62
  - single resource routing, 62–63
  - update action, 62
  - updating users, 62
  - user accounts, creating, 62
  - user profiles, displaying, 62
  - users, listing, 61
- StartWorkerThread boot loader, 31

- Status codes
    - rendering templates, setting, 94
    - requests, sending, 73
    - in testing, 291–292
  - status** method, 73
  - status=** method, 73
  - :status** option, 94
  - Stepping through a list of values, 176–177
  - Storage mechanisms
    - cookie sessions, 211–214
    - CookieSession** class, 211–214
    - DataMapper sessions, 217–219
    - garbage collection, 215–216
    - memcached sessions, 216–217
    - memory sessions, 214–216
    - MemorySessionStore** class, 215–216
    - reaping sessions, 215–216
    - TamperedWithCookie** error, 214
  - Store containers, 208–211, 214–215
  - Storing sessions. *See* Sessions, storing.
  - Strategic eager loading, 150–152
  - strategies.rb** file, 222
  - Strategy** class, 227–231
  - Strategy stores, 270
  - stream\_file** method, 74
  - Streaming files, 74, 97–98
  - String** class, 21, 128
  - Strings
    - camel case, converting to, 21
    - encrypting, 155
    - escaping special characters, 21, 155
    - globs, 29
    - as inline templates, 23
    - joining in file paths, 21
    - language translation, 21
    - parsing as CSVs, 155
    - path/constant conversion, 21
    - snake case, converting to, 21
    - storing, 128
    - truncating, 167–168
    - in URLs, 155
  - String.translate** method, 21
  - String.translations** method, 21
  - Stub classes, setting up, 31
  - Subclasses, listing, 69
  - subclasses\_list** method, 69
  - sum** method, 159
  - Summing numerical properties, 159
  - swift** key, 25
  - Swiftiplied Mongrel adapter, 25
  - Symbolic matching
    - routing match rules
      - ( ) (parentheses), in optional matching, 52
      - automatic parameters, 51
      - flexible segmentation, 52
      - optional matching, 52–53
      - overview, 51
      - segment-specific regular expressions, 52
  - Symbolic segments, 40–41
  - Symbolized class names, 136
  - Symbols
    - registering routes with, 56–57
    - thrown content, catching, 71
- ## T
- t** method, 21
  - Table keys, defining, 131
  - Tag helpers, 177–178
  - tag** method, 178
  - Tags, Haml views, 105–106
  - TamperedWithCookie** error, 214
  - Template engines, init script file, 14
  - template\_roots** method, 70, 71
  - template\_roots=** method, 70, 71
  - Templates. *See also* Views.
    - capture** method, calling, 71
    - capturing ERB blocks, 71

Templates. *See also* Views. (*continued*)

- `concat` method, calling, 71
- concatenating ERB blocks, 71
- embedding in templates, 71
- ERB blocks, 71
- inline, strings as, 23
- inlining, 29–30
- mailers, 261
- Merb view templates, 109–111
- reloading, 32
- rendering. *See* Rendering, templates.
- root location, 70, 71
- within templates, 111–112
- templating system, default, 101–102
- throwing content, 71
- thrown content, checking for, 71
- view. *See* Flat application layouts.

Templates boot loader, 29–30

Testing

- `be_client_error` matcher, 291–292
- `be_missing` matcher, 291–292
- `be_successful` matcher, 291–292
- coverage task, 282
- extensions, 293–294
- `have_body` matcher, 292
- `have_content_type` matcher, 292
- html task, 282
- Kernel methods, 293–294
- mailers, 261
- model specs, 283–285
- model task, 281
- rake tasks, 279–282
- `redirect` matcher, 292
- `redirect_to` matcher, 292
- request helper, 289–290
- request matchers, 290–292
- request specs, 285–288
- request task, 281
- `respond_successfully` matcher, 291–292

- routers, URL generation and recognition, 49–50

- RSpec extensions, 292–294
    - spec files, 282–283
    - `spectasks.rb` file, 279–280
    - status codes, 291–292

Testing environment, applications, 7, 16–17

Testing options, init script file, 15

`:test_method` method, 261

`test.rb` files, 17

`test_send` method, 254

`Text` class, 128

Text data, storing, 128

Text MIME set, rendering views with, 71

`text_area` method, 189–190

`text_field` method, 189–190

Thin adapter, 25

`thin` key, 25

`through` method, 138–139

`throw_content` method, 71

`throw_content?` method, 71

Tilde (~), preserving whitespace, 108

Time, returning, 22

`Time` class, 22, 128

Time DSL, 173–175

`time` format, 171

`time_lost_in_words` method, 175

Timestamps, 156–158

`to` method, registering routes with, 54–55.

*See also* `Register` method; `With` method.

`to_const_path` method, 21

`to_const_string` method, 21

`to_currency` method, 169–170

`to_datetime` method, 22

`to_json` method, 22

`to_time` method, 22

`trace_log!` method, 11

`:track` option, 132

`TrueClass` class, 128. *See also* `Boolean` class.

`truncate` method, 167–168

`try_dup` method, 21

`two-digits` method, 168

## U

Underscored filters, 248

`unescape_regex` method, 21

Unified class and instance variables,  
creating, 20

Uniform Resource Locators (URLs). *See* URLs  
(Uniform Resource Locators).

`unique` query parameter, 146

`:unique_index` option, 132

Uniqueness, validating, 162–163

`:unless` option, 83

`update` action, 62, 248

`update_attributes` method, 152–153

`update_bound_check_box` method,  
183–184

`update_bound_controls` method,  
186

`update_*_controls` method, 181–182

`update_speed` method, 153

Updating

attributes, 152–153

records

`attribute_set` method, 152

original values *versus* dirty values, 153

overview, 152

`save` method, 152

`update_attributes` method,  
152–153

`update_speed` method, 153

updating attributes, 152–153

users, 62

`upgrade_model_storage` method,  
123–126

URI property type, 155

URIs, storing as strings, 155

`url` method

`AbstractController` class, 71

Merb console, 11

URLs, generating, 11, 50

URL methods, 71

URLs (Uniform Resource Locators)

absolute, returning, 71

deleting, 11

generating, 11

getting, 11

posting, 11

putting, 11

relative, returning, 71

retrieving, 11

router generation, testing, 50

router recognition, testing, 49–50

strings in, 155

User accounts

creating, 62

hijacking, 59–60

`user_class` method, 231

Users

deleting, 62

listing, 61

profiles, displaying, 62

updating, 62

`use_template_engines` method, 14

UUID property type, 155

UUIDs, storing as strings, 155

## V

`validates_absent` method, 162

`validates_format` method, 162

`validates_is_accepted`  
method, 162

`validates_is_confirmed`  
method, 162

`validates_is_number` method, 162

`validates_is_unique` method,  
     162–163  
`validates_length` method, 162  
`validates_present` method, 162  
`validates_with_block` method,  
     162  
`validates_within` method, 163  
`validates_with_method`  
     method, 162  
 Validations  
     automatic, 132  
     automatic, turning off, 132, 163  
     with blocks, 162  
     conditions, 163–164  
     confirming attributes, 162  
     contexts, 164  
     errors, 164–165, 186, 191  
     format of a value, 162  
     formats, 163  
     with methods, 162  
     numerical length, 162–163  
     numerical values, 162  
     overview, 162–163  
     presence of, 163  
     presence of a value, 162  
     within a range of values, 162–163  
     against regular expressions, 132  
     true values, 162  
     uniqueness, 162–163  
     `validates_within` method, 163  
 Very flat application layouts, 3–6  
`very_flat` argument, 3–6  
 View methods, 71  
 View templates. *See* Flat application layouts.  
 Views. *See also* Templates.  
     authentication, 249–250  
     for ERB. *See* ERB (eRuby) views.

    for HTML. *See* Haml views.  
     login, 249–250  
     Merb view templates, 109–111  
     partials, 111–112  
     templates within templates, 111–112  
     templating system, default, 101–102  
     for XML. *See* Haml views.  
 VirtualFile class, 23

## W

Warn messages, logging, 18  
 WebController mixin, 265–266  
 WEBrick adapter, 25  
`webrick` key, 25  
 Weizenbaum, Nathan, 105  
 Whitespace  
     preserving, 108  
     removing, 102  
     sensitivity, 19  
 Widgets. *See* Parts.  
`with` method, registering routes with, 55.  
     *See also* `Register` method; `To`  
         method.  
`:with` option, 83  
 Worker queue, placing blocks in, 74  
 Worker threads, starting, 31  
 Workers, definition, 34  
`writable?` method, 271–272  
`write` method, 271–272  
`:writer` option, 131  
 Writing cache data, 271–272

## X

XML entities, escaping, 74  
 XML MIME set, rendering views with, 71

**Y**

## YAML

- configuration file, 114–115
- files, 19–20

- MIME set, rendering views with, 71
- nodes, 115
  - storing as a string, 155
- `Yaml` property type, 155