

# Merb in Action

Michael D. Ivey  
Yehuda Katz  
Ezra Zygmontowicz



 MANNING



Unedited Draft



**MEAP Edition  
Manning Early Access Program**

Copyright 2008 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

Please post comments or corrections to the Author Online forum:  
<http://www.manning-sandbox.com/forum.jspa?forumID=450>

## **Table of Contents**

### **Part 1: Learning Merb**

- 1 Introducing Merb**
- 2 Getting Started With Merb**
- 3 Using merb-gen To Build Your App**
- 4 Running the Application**

### **Part 2: Using Merb**

- 5 Testing Merb Applications**
- 6 ORM Plugins**
- 7 Mailers**
- 8 Parts**
- 9 Deployment**

### **Part 3: Hacking Merb**

- 10 Request Lifecycle**
- 11 merb-core Components**
- 12 merb-more**
- 13 ORM internals**

# 1

## Introducing Merb

In the world of Web 2.0 (and with shiny new Web 3.11 just around the corner), frameworks for building database-driven web applications are easy to find in your language of choice. In Ruby alone there's Rails, Camping, Sinatra, Waves, Nitro, Ramaze, IOWA ... and the list goes on. Another of the Ruby frameworks is Merb. Fast, flexible, and very accessible, Merb describes itself as "All you need. None you don't." Merb comes with everything you need to build fast, scalable web applications (and then some), and yet maintains a very high level of flexibility and configurability. Developers choose frameworks for many reasons, but many are drawn to Merb initially because it offers them flexibility in how to approach their applications.

While most use Merb in a traditional MVC style, Merb itself allows many approaches and encourages developers to adapt the framework to their application's needs. Since Merb grew up alongside Ruby on Rails, there's a strong temptation to compare the two. We'll try to avoid explaining Merb in terms of Rails after this chapter. Where it would be helpful, however, we'll call out features of Merb that are implemented differently in Rails, to help ease the transition for those coming from a Rails background. Before we can understand why Merb matters and why it works the way it does, let's first take a look at the state of things around Rails version 1.1.5.

### 1.1 Historical Context

Officially announced in October 2006, Merb started life as a small hack. Designed by Ezra Zygmontowicz to work alongside Ruby on Rails applications, Merb was intended for tasks, like file uploads, where the Ruby on Rails framework underperformed. Just months earlier, Apple had announced that it would include Rails with the next version of its flagship operating system, OSX Leopard, a sign of the framework's maturity. In the years since, Merb has found *itself* maturing from a side-hack to a full-fledged framework in its own right. As Merb continued to attract users and contributors, people began using it for more than just supplementing existing Rails applications.

#### NOTE

To see how simple Merb was at the beginning, check out the entire source in less than 120 lines:  
<http://pastie.caboo.se/14416>

Instead of focusing on making it easy for brand-new programmers to put together a blog or shopping cart in minutes, Merb found itself focusing on modularity, efficiency, speed, and thread-safety. But since Merb took its basic architecture from Ruby on Rails, it shares much of the ease of use that brought so many new web programmers to Ruby in the first place.

As the community continued to grow, much of the ease of use that was missing in earlier versions of Merb was added in to both the core framework and as plugins. Because Merb values modularity through plugins, that meant that most of the new features forced the framework designers to think intelligently about keeping the core of the code light and easy to extend. In fact, much of what makes up Merb is implemented as plugins on top of the core, but you'll hardly notice unless you start digging in yourself. We'll cover a lot of those details in Part II, when we crack open the hood of Merb and take a look at how it all fits together.

Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=450>

## 1.2 Ruby

Popular with a devoted group of Japanese programmers since its inception in the 1990s, Ruby did not attract a lot of attention in the United States until about ten years later. Yukihiro Matsumoto (affectionately known as "Matz") designed Ruby as a language that focused on developer joy and ease of use. Ruby programs are often evaluated not only by how well they perform or meet the design specifications but how aesthetically pleasing the code is. Some Ruby programmers compare the process of writing code to the process of writing poetry. Newcomers to Ruby are often surprised and delighted to find an open and helpful attitude present in Ruby forums and chat rooms. They are experiencing a defining characteristic of the Ruby Community - the people involved are just plain nice. This attitude is not a spontaneous occurrence, it has been cultivated since the early days of the language. This principle of niceness has its own acronym, MINASWAN, which stands for "Matz Is Nice and So We Are Nice".

With a language focussed on beauty and programmer happiness, and a friendly and helpful community, it's no wonder that we love Ruby as much as we do. Throughout the book, we assume a certain level of comfort with the Ruby Programming Language. For readers new to Ruby, the appendix contains a basic introduction to Ruby concepts to help with the transition to Merb.

## 1.3 Why Another MVC Framework?

With the existence of Ruby on Rails, Camping, Waves, and Nitro, you might be asking yourself: Why another framework? And in particular, why another framework that uses the Model, View, Controller paradigm when Rails is so popular, successful and easy to use.

### NOTE

Model, View, Controller is a style of programming that emphasizes breaking your code into three components. The models control access and control data sources. The Views represent the visual representation of the data sent to the user. Finally, the controller stitches the two together, passing useful model data to the views. In the case of web frameworks, the models typically represent access to a database, the views are HTML templates, and the controllers route requests to their appropriate templates.

For many programmers, Rails solves all problems. It provides a useful abstraction that was originally extracted from a real application, BaseCamp, and that provides a toolkit that can get people up and running quickly. However, that ease of use is quickly supplanted by the opinionated nature of the framework. In order to simplify the decisions you'll have to make while using Rails, its framework designers make many decisions for you. That means that overriding those defaults can begin to consume most of a Rails programmer's time. It is said that any non-trivial Rails application involves hacking on the core framework, and your mileage may vary.

On the other hand, Merb is designed around the notion of modularity, to the extent that the core framework itself is designed as a group of modules. Because the framework designers need to keep hooks open for the framework itself, you can be sure that you'll be able to override small and large pieces of functionality as necessary. In other words, Merb is designed to be hackable. Designed from the ground-up with efficiency in mind, Merb is also substantially faster than its counterparts, being able to spit out an impressive 2,000 requests per second for simple strings, and more than 1,500 requests per second when using templates. At time of press, it is consistently two to five times faster than Rails, with higher benefits coming from applications that make heavy use of partials. In fact, part of Merb's release process is a comprehensive benchmark. If a release of Merb runs slower than the previous release, the core team tweaks it to beat the previous release.

Merb also maintains a healthy agnosticism toward companion libraries. For instance, it works out of the box with both `Test::Unit` and `rspec`, works with any of the `ActiveRecord`, `DataMapper`, and `Sequel` ORMs, and works with any of the myriad JavaScript libraries out there. Merb achieves this by providing basic hooks into the framework, and allowing plugins (like `merb_test_unit`, `merb_activerecord`, etc.) to provide functionality appropriate to the component in question. This goes as far as to generate the appropriate type of stub during model generation, providing a simple mechanism for using Merb that shapes itself around your choices. Now, let's take a closer look at what these core elements of Merb's philosophy mean in practice.

### 1.3.1 Hackability

We'll take a more complete look at exactly how clean and modular the Merb framework is in Chapter 3, but suffice it to say that its designers built it with hacking in mind.

Let's take a look at a simple example involving Merb's Controllers and Mailers. Instead of implementing web controllers and mailers separately, with web controllers containing all the logic for handling requests and responses, and mailers containing the logic for sending mail, Merb takes a more modular approach. Merb defines a super-class, `Merb::AbstractController`, which contains all the logic for helpers, layouts, before and after filters, and simple dispatch. `Merb::Controller` inherits from the `AbstractController`. A separate module, called `merb-mailer`, defines a `Merb::MailController` which also inherits from `AbstractController`. This is an immediate win for Merb, because users of the framework can leverage their knowledge of web controllers to write DRY and modular mailers with all of the features of web controllers. And because both modules inherit from the same module, improvements to the general module automatically percolate down to all inherited modules.

#### NOTE

Ruby on Rails has two separate classes, `ActionController` and `ActionMailer`, which are implemented separately and have two different general APIs. While `ActionController` supports layouts, helpers and instance variable assignment, `ActionMailer` is much simpler, with none of those features.

Because of this clean approach, it's very easy to override the existing behavior or define new controller types. For instance, another optional module, `merb-parts`, provides support for simple, light, reusable components with full support for helpers, filters, layouts, and content-negotiation. Merb is also built on top of the Rack interface, which provides a single, unified interface for web servers. This allows Merb to easily support any web server including Mongrel, Evented Mongrel, Thin, FCGI, CGI, and even Webrick. And because the Rack interface is so simple, Merb's console mode is implemented as a Rack stub in under 50 lines of code. As a result, understanding and hooking into Merb's web server interface is trivial. And get this: Rack's interface allows you to mount multiple applications, which it will try one at a time until one of them does not return a 404. In a later chapter, we'll show you how to use this feature to mount simple Rack applications to handle things like RSS feed generation, so you can still use Ruby, but skip the overhead of Merb for cases that don't need the entire framework. Its features like this that can help you build more efficient apps.

### 1.3.2 Performance

Designed with performance in mind, Merb has surpassed other Ruby web frameworks, including Rails by leaps and bounds. While Ruby apps are capable of "scaling" just fine by adding additional hardware, the Ruby language can sometimes make it easy to introduce features that prevent efficient scaling. And while it has become conventional wisdom by now that Rails apps can scale by simply throwing hardware at increased demand, the central argument is flawed. People who have come to Ruby have been convinced that since human developers are more unique (and more expensive) than hardware, it's worthwhile to use frameworks that trade development time for hardware costs.

While that might be true if the tradeoff was so neat, it introduces a false dilemma. As Merb has shown, it's possible to build a framework that can scale efficiently while still maintaining simplicity for developers. In real-life terms, a framework that can serve twice as many pages per second requires half as many servers. And the mere fact of needing to introduce additional boxes to handle scaling adds complexity to a deployment, which can eat into the developer savings.

Merb provides performance and scaling that allow you to benefit from the developer ease of Ruby while still maintaining respectable performance stats. Also, because it's a Ruby app, you get the same general deployment and scaling approach as other Ruby frameworks. In other words, the work the community has done to solve the general issues in this area is directly applicable to Merb, and we'll discuss that in much greater depth in Chapter 10, Deployment.

### 1.3.3 Agnosticism

As the Rails community matured, and the framework found itself in more varied kinds of applications, developers found they didn't always agree with the opinionated choices Rails makes. Lots of developers started to use `spec`,

Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=450>

an alternative testing framework to Rails' built-in support for Test::Unit. New ORMs, including DataMapper and Sequel, came out of the woodwork and began to flourish. And for JavaScript, many Rails developers use jQuery or MooTools rather than the built-in support for Prototype and Scriptaculous.

The problem is, Rails prides itself on its opinionated nature and batteries-included packaging. As a result, users of rspec find themselves out of luck when they generate new controllers (Rails, by default, creates Test::Unit stubs). While the creators of rspec eventually created packages to hook deep into Rails and replace the built-in support for Test::Unit, they use interfaces that are subject to breakage with new versions. Similarly, dropping in a new ORM, like DataMapper is possible in Rails only if the creators of the ORM make special efforts to support Rails. As more plugins hook deeper into Rails, the larger the chance of conflicts and breakage. And obviously, many Rails plugins make assumptions about things like template languages. For instance, plugins designed to use Rails' built-in ERB template language may not work correctly with Haml, Markaby, or Liquid. Merb, on the other hand, takes special care to provide hooks into the base framework, instead of making opinionated assumptions about what collection of tools our users will use. By default, Merb uses Erubis, Rspec, DataMapper, and jQuery (so new users can get a full, batteries-included stack), but nothing in the core framework assumes that stack. In fact, with the exception of Erubis, the "default" Merb stack is simply a set of plugins designed, maintained, and cared for by the Merb core team.

Here's an example of how Merb implements templating, to keep things agnostic: template languages are required to expose a single method (`compile_template`), which takes the path of the template, the name of the method that will be compiled, and the module it will be compiled into. Don't worry about the details; we'll dig into the inner workings of Merb in Part III. The point, however, is that Merb provides very simple points of interaction with other modules, assuming very little so that you can drop in new template languages, ORMs, testing frameworks, or JavaScript libraries with no difficulty at all.

### **1.3.4 Yikes!**

At this point, you might be thinking that Merb forces you to make all sorts of decisions yourself. In the name of modularity, efficiency, and hackability, it seems, we're throwing new developers under the bus. Don't worry, new Merbivorian.

The Merb maintainers understand that not everyone's ready to dive in and configure every little piece of their new framework. While there might come a day where being able to fine-tune your app becomes important, you can start your journey with a Batteries Included version of Merb, which includes `merb-core` (the core framework) as well as `merb-more` (generators, mailers, parts, alternate template languages, etc.). From your perspective, you'll just need to do `gem install merb`, and for now, not need to worry about how everything fits together. The core team also maintains a series of high-quality plugins, including plugins for Ruby ORMs, support for Test::Unit, and some additional helpers that are not included in `merb-more`. You can see how everything fits together in figure 1.1.

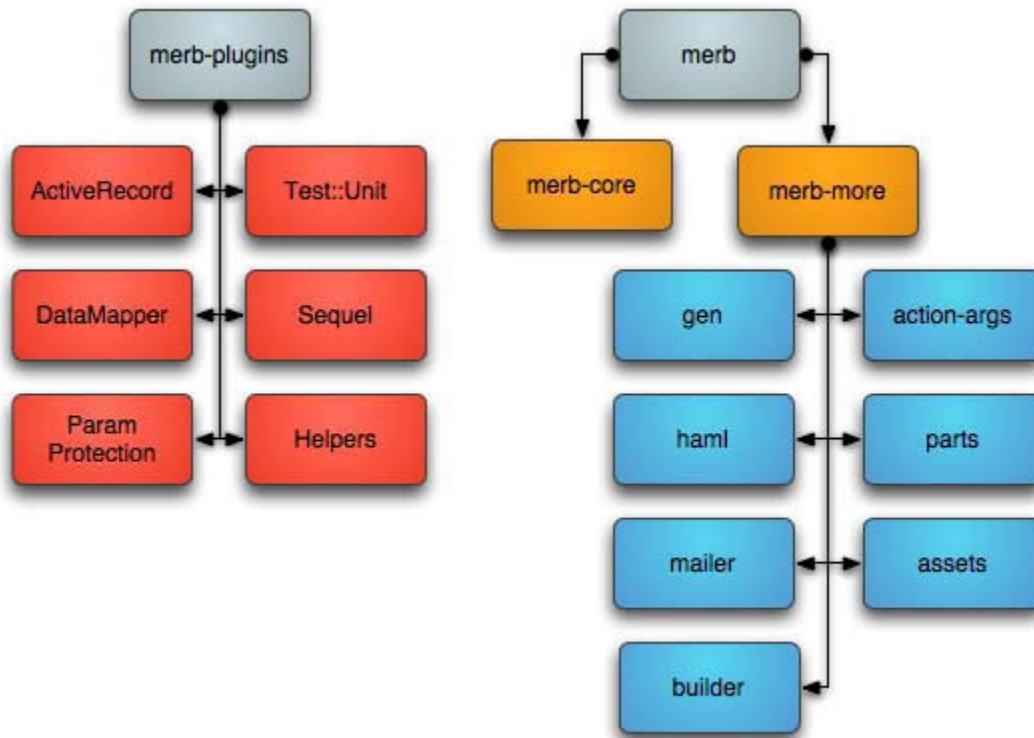


Figure 1.1 Merb components

As we move forward, we'll highlight what component part of Merb contains each feature we discuss.

## 1.4 A Taste of Merb

Nothing gives a better taste of a framework than code. Over the next few pages we're going to show you some pieces of Merb applications. If this feels rushed, that's because it is. These examples aren't designed to teach you how to do these steps. We just want to give you a general feeling for Merb. Where appropriate, we'll note the chapters or pages you can reference for more information.

### 1.4.1 What You Return Is What You Get

Whether it's Ruby or Java or C#, whether it's a Web framework or a GUI toolkit, with many frameworks you end up writing code that doesn't match the idioms of the language. Your application looks like others written with that framework, but may not "feel like Ruby".

Merb's controllers try to be good Ruby citizens. When you instantiate one it is in a consistent state. All of the controller's dependencies are passed in via the constructor. Most of the methods are public. In short, Merb controllers act like Ruby objects instead of pieces of the framework. Because of this, Merb controllers are easy to test and they're easy to re-use. Here's how you might create a new controller instance in a test.

```
@controller = Cats.new(@mock_request)
@controller._dispatch("index")
```

Figure 1.2 Creating a controller and calling an action

That's all there is. (You can't call the action directly with `@controller.index` because Merb runs filters before and after the action method.) Merb is very tolerant when it comes to what gets passed in as the request, too: as long as it acts like a `Merb::Request`, you can pass in a mock or a test stub.

Now let's look inside that example controller in figure 1.3 and see how the response body is generated. This is another place where Merb's simplicity is sure to make Ruby programmers feel at home.

```
class Cats < Application
  def index
    "I know about these cats: " + find_all_cats.join(", ")
  end
end
```

Figure 1.3 Returning a response

Merb uses the return value of the action as the response body. This means that whatever data you would get from calling `@controller._dispatch("index")` is the same data that will be sent to the web browser. Merb controllers behave like other Ruby objects would.

Of course, we need more than just strings, and no one wants to generate HTML by hand, so let's look at just one of Merb's options for rendering templates.

### 1.4.2 Templates and Web Services

Chances are you've either built a web service, or thought about building one. With Web 2.0 data exchange is on everyone's mind. Merb gives you some simple and powerful tools for adding RESTful web service support alongside a traditional application.

#### NOTE

Representational State Transfer (REST) is an architectural style for building applications, originally explained by Roy Fielding. See [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) for his original paper.

REST also describes an implementation of that architectural style that maps HTTP verbs to standard actions, as implemented in Rails. Merb supports Rails-style REST and more general RESTful concepts.

Let's return to our application that manages a database of cats. This time, we'll assume we've done a lot more on it, and have an index and show action, both working with our models and rendering HTML.

```
class Cats < Application
  def index
    @cats = Cat.find_all
    display @cats
  end

  def show
    @cat = Cat.find(params[:cat_id])
    display @cat
  end
end
```

Figure 1.4 Cats controller

We'll go into much more detail on the `display()` method in chapter XX. For now, just know that if it finds a template for the action and the current content type, it will render it. Otherwise, it will try and coerce the object into that content type.

Since we haven't changed any of the standard settings, this controller only supports HTML output. Suddenly we realize that our cat database needs XML support, so we can share data with a hot new pet aggregator. Because Merb is ready to handle multiple content types, we only have to add one line.

```
class Cats < Application
  provides :xml

  # |...

end
```

Figure 1.5 Adding XML support

Once we tell Merb that our controller handles XML, we're done. (Mostly. There's a little more to the story, which we'll cover later, but for this high-level picture, it really is that simple.)

### 1.4.3 When Things Go Wrong

We can't expect everything to always go our way. Users give us bad data, databases crash, things fall apart. In Ruby, when things go wrong, we're used to raising exceptions. Merb brings that philosophy to dealing with error handling inside controllers.

Let's handle the case where the cat the user asked for doesn't exist. Our show action from the previous section would look like figure 1.6.

```
class Cats < Application
  def show
    @cat = Cat.find(params[:cat_id])
    raise NotFound unless @cat
    display @cat
  end
end
```

Figure 1.6 Cats#show with error handling

Now, if the cat record isn't found, the end-user will see a "404 Not Found" error. Merb defines custom exceptions for each of the HTTP status codes. If the HTTP spec ever adds payment rules, you can `raise PaymentRequired` as part of your business plan.

### 1.4.4 Web Parameters and Method Arguments

There will always be a little bit of a disconnect between application programming and the web, but Merb has a few ways to lessen it. Parameterized actions (part of *merb-more* in *merb-action-args*) makes your actions look like traditional Ruby methods, including default values for optional arguments. Compare figure 1.7 with figure 1.6.

```
class Cats < Application
  def show(id)
    @cat = Cat.find(id)
    raise NotFound unless @cat
    display @cat
  end
end
```

Figure 1.7 Cats#show with action args

## 1.5 When to Use Merb

When should you use Merb? Whenever possible, we hope! While Merb is primarily appropriate for building web applications, that's not even a real limitation (see future note/example re: non-web Merb app). Inside merb-core you'll find everything you need to build simple web applications, including basic authentication and X. Most Merb applications involve a database and take full advantage of the various ORM plugins. Merb shines for non-database driven applications, though. Imagine an API that aggregates 3 different feeds into a unified format. Because you only include what you need, there's no ORM or database layer involved at all. Merb is especially well-suited for building APIs where there isn't a web front-end, using the web as a transport layer. Because you can use just merb-core, you can strip out everything you don't need, and build a small, light-weight application. Another application that's common is using Merb to upload files into an existing Rails application.

When shouldn't you use Merb? If you want transparent Javascript updates generated by your Ruby code, you'll notice that Merb doesn't offer any of that, by design, and instead encourages you to use the methods described as "Unobtrusive Javascript". If you're looking for the same level of ease and simplicity as you find in Rails, Merb may not make you happy. Merb trades some of the simplicity and convenience for flexibility.

## 1.6 Summary

Merb was created out of a belief that Ruby on Rails has a number of excellent ideas, but that it can be sorely lacking in implementation, especially in the core areas on hackability, performance and vendor lockin. Merb is built around a lot of the same core concepts, like MVC architecture, convention over configuration, and a batteries-included stack, but makes it much simpler to override and swap out parts of the stack that aren't working. In the next few chapters, we'll get you up and running with Merb. We think you'll be pleased with how simple getting started can be, and if you've used Rails, you're sure to experience a pleasant sense of déjà vu. That's because many of the most pleasing parts of getting up and running with Rails are almost identical in Merb. Let's go!