**Taking Merb for a Spin**

May 31st, 2007 • [Merb](http://depixelate.com/merb)

Looking for an excuse to check out [Merb](http://merb.devjavu.com)? I was too and finally had a chance to check it out on the plane to RailsConf. The code base is definitely a work in progress but the code is very impressive and approachable. I'd encourage everyone to take a look at it - especially the ultra clean routing code which rocks the house!

Interested in taking a test drive? Good, keep reading.

**Install Merb**

$ sudo gem install merb

**Sample Invoice Tracker app**

With your forthcoming Merb hacking skills likely to result in a landslide of consulting gigs, we better build a custom app to track clients and invoices. If you want to skip ahead and grab the code you find it [here](http://svn.depixelate.com/applications/invoice_tracker).

$ merb -g invoice\_tracker

Merb-gen app invoice\_tracker

The -g flag tells merb to create a skeleton app.

You can checkout all the options with the -h flag.

$ merb --help

**Layout**

The standard Merb layout is very similar to a Rails app except all code is in a dist subfolder. This makes sense as all you need to deploy is the dist folder when it comes time to put the app into production.

The rest of the layout should be very familiar for a Railer.

**Database Config**

Database setup currently occurs in *conf/merb\_init.rb*. This is one part of Merb that could use some refactoring. It would be nicer to externalize this specification into a yaml file where multiple environments could be specified.

For now, put in your credentials. Don't forget to create the database too...

**Routing**

*conf/router.rb* is where routes are specified.

Merb::RouteMatcher.prepare do |r|

# r.default\_routes installs stuff like

#

# /:controller/:action/:id

# /:controller/:action

# /:controller, :action => 'index'

# ...

#

r.default\_routes

# just like Rails :)

r.resources :clients

end

For this simple app all we need is:

**conf/router.rb**

Merb::Router.prepare do |r|

r.resources :clients do |client|

client.resources :invoices

end

r.add '', :controller => 'clients', :action => 'index'

end

**Migrations**

Ok, time to bang out this little app so we can start billing hours!

First let's add a new migration:

$ ./script/new\_migration CreateClients

$ ./script/new\_migration CreateInvoices

Now check out the schema/migrations directory. Notice that 001\_add\_sessions\_table.rb is the first migration which was automatically created by the merb generator. We also see our new migrations named 002\_create\_clients.rb and 003\_create\_invoices.rb.

Let's create two simple migrations:

**schema/migrations/002\_create\_clients.rb**

class CreateClients < ActiveRecord::Migration

def self.up

create\_table :clients do |t|

t.column :name, :string, :null => false

t.column :address, :string

t.column :city, :string

t.column :state, :string

t.column :zip\_code, :string

end

end

def self.down

drop\_table :clients

end

end

**schema/migrations/003\_create\_invoices.rb**

class CreateInvoices < ActiveRecord::Migration

def self.up

create\_table :invoices do |t|

t.column :client\_id, :integer, :null => false

t.column :description, :string

t.column :invoiced\_on, :datetime, :null => false

t.column :rate, :decimal, :null => false, :precision => 9, :scale => 2

t.column :hours, :decimal, :null => false, :precision => 9, :scale => 2

end

end

def self.down

drop\_table :invoices

end

end

Now let's run those migrations:

$ rake db:migrate

**Models**

As you may have already guessed ActiveRecord is the standard Merb way here so it's the same way Rails does it.

**app/models/client.rb**

class Client < ActiveRecord::Base

has\_many :invoices, :dependent => :destroy, :order => :invoiced\_on

validates\_presence\_of :name

def total\_invoiced

@total\_invoiced ||= self.invoices.inject(0){ |sum, i| sum += i.total }

end

end

**app/models/invoice.rb**

class Invoice < ActiveRecord::Base

belongs\_to :client

validates\_presence\_of :rate, :hours, :invoiced\_on

def total

self.rate \* self.hours

end

end

**Controllers**

Let's create our clients controller with one initial index action. Notice that you leave off the \_controller suffix preferred by Rails and must explicitly call render at the end of actions.

**app/controllers/clients.rb**

class Clients < Application

def index

@clients = Client.find(:all)

render

end

end

**Views**

Views are again very similar to Rails. You can use the following view suffixes: herb, jerb, erb, and rhtml. Let's create a placeholder view for initial testing.

**app/views/clients/index.erb**

<h1>Clients</h1>

**Run the App**

$ merb

Navigate to http://localhost:4000

Hopefully it's all working and you are looking at a page that says "Clients"

See, this Merb stuff isn't tough at all!

Let's take a look at a few other features...

**Filters and Mime type handing**

Let's look at the fleshed out clients controller:

class Clients < Application

before :find\_client, :exclude => [ :new, :create, :index ]

def new

@client = Client.new

render

end

def create

@client = Client.create(params[:client])

redirect clients\_path

end

def index

@clients = Client.find(:all)

respond\_to do |format|

format.html { render }

format.xml { render :xml => @clients.to\_xml }

end

end

def show

respond\_to do |format|

format.html { render }

format.xml { render :xml => @client.to\_xml }

end

end

def edit

render

end

def update

@client.update\_attributes(params[:client])

redirect client\_path(@client)

end

def destroy

@client.destroy

render\_js

end

private

def find\_client

@client = Client.find(params[:id])

end

end

Notice that Merb has the *before* "macro" to implement filters that run before actions. The cool thing about Merb filters is how they signal to halt the action. In Rails the returning of false to halt filters is rather arbitrary. It works but it's not elegant. A Merb filter thows the symbol :halt. Let's take a look at how this could work:

Let's suppose we want to limit visibility of certain actions to admins only. We can use a before filter like this

class Clients < Application

before :authorize

...

private

def authorize

throw(:halt, 'No way Jose!') if !current\_user.admin?

end

end

This syntax is clean and clearly states the intention of the filter. Use :exclude and :only options to designate to which actions the filter applies.

Mime type handling is done in a very similar way to Rails with respond\_to blocks. The few differences are:

* Requirement of explicitly calling render
* Calling render\_js to render jerb templates (sets content type header to text/javascript)
* No named urls yet (the redirects above that appear to be named routes are actually helper methods)

**Form stuff**

Merb has only a few of the niceties provided by ActiveSupport. We can however create form elements that map to ActiveRecord Models using the *control\_for* methods:

<form action="/clients" method="post">

<%= control\_for @client, :name, :text %>

<input type="submit" value="Create Client" />

</form>

**Debugging and Patching**

Want to add some logging statements or tweak Merb a bit? No problem. Merb has a rake task that freezes the framework into the dist/ folder.

$ rake merb:freeze

Now you tweak the code and then refreeze to wipe out any unwanted changes. Found a bug and looking to make a patch. Just freeze the framework from subversion.

$ rake merb:freeze\_from\_svn

Once the framework is frozen, use the following server command to bootstrap Merb with the local version:

$ ./script/merb

**Helpers**

Again, helpers are similar to those in Rails. Throw all global helper methods in *app/helpers/global\_helpers.rb*. For controller-specific helpers, create a file named after the controller in the *app/helpers* dir.

**app/helpers/clients\_helpers.rb**

module Merb::ClientsHelper

def print\_stuff

'stuff'

end

end

**Layouts**

Again Merb follows the general semantics of Rails when it comes to layouts. Merb will look for a layout named after the current controller and fallback to application.herb if one does not exist. There are also various helpers in Merb::ViewContextMixin like require\_css and js\_include\_tag that are useful in layouts.

**Sample Invoice Tracker app**

1. Grab the code
2. $ svn co http://svn.depixelate.com/applications/invoice\_tracker
3. $ cd invoice\_tracker/
4. Create database named 'invoice\_tracker'
5. Update username and password in dist/conf/merb\_init.rb
6. Run migrations
7. $ rake db:migrate
8. Start Merb
9. $ merb
10. Play with app at http://localhost:4000

**Final thoughts**

I hope I've wetted your appetite to install Merb and poke around. Ezra has really put together a great little framework and I highly recommend you take closer look. The code is very readable and the small size of the codebase makes it a quick study.

There is also a sample blog app called [Mrblog](http://svn.devjavu.com/merb/mrblog/trunk) that you can look at to learn a bit more.

Important: Note that Merb changes frequently as Ezra and contributors enhance and patch up the codebase. If you're looking for help, you can post to the [Mailing list](http://rubyforge.org/mailman/listinfo/merb-devel) or troubleshoot directly from the code.

**Update 5/30/07**

Ezra just released Merb 0.3.3 which has both a small router specification change and route generation helpers. I updated the sample app to work with the new routing format so if you happen to have Merb already installed, please upgrade to the latest and greatest.

**References**

* [Merb](http://merb.devjavu.com)
* [Mailing list](http://rubyforge.org/mailman/listinfo/merb-devel)
* [Mrblog](http://svn.devjavu.com/merb/mrblog/trunk) example app

--- --- ---

**5 Comments**

1. Comment by [*Olle Jonsson*](http://ollehost.dk/blog) on 05/31/07

Thanks many times for making a sample app, Zack. This paves the way for more Merb usage. Excellent job. Having so many smaller choices, below the Rails "giant", we need good docs and samples to look at, to be able to imagine what kinds of apps are good with each particular mini-framework. You just furthered that effort.

1. Comment by [*Ezra Zygmuntowicz*](http://brainspl.at) on 05/31/07

Thanks for the writeup Zack! I actually just released merb0.3.3 gem an hour ago. It has a route generator now as well as some nice enhancements in rendering and some other refactors. you need to use Merb::Router.prepare instead of Merb::RouteMatcher.prepare in your router.rb file. The merb app generator sets this correctly now so its only if your upgrading. People with questions should hop onto irc.freenode.net and /join #merb . I'm usually in there.

1. Comment by *Ralf Hessmann* on 05/31/07

Many Thanks for this tutorial, Zack !!! One change, see post from Ezra: http://brainspl.at/articles/2007/05/31/new-merb-tutorial-and-restful-sample-application "... There is one slight change in the router.rb route definition that you have to change if you are upgrading from an older merb. In dist/conf/router.rb, you need to use Merb::Router.prepare instead of Merb::RouteMatcher.prepare."

1. Comment by [*Zack*](http://depixelate.com) on 05/31/07

Thanks Ezra and Ralf: I posted a new version of the sample app to reflect the router.rb route definition change.

1. Comment by *Kevin* on 05/31/07

Hmm, would like to get some merb on, but for some odd ready hitting the page in Safari downloads the page rather than displaying it in the browser. Works A-OK in Firefox. Any ideas?