

# URL Mapping with Horde/Routes

php|works 2008  
Mike Naberezny



Maintainable  
Software

<http://maintainable.com>

# About Me

- <http://mikenaberezny.com>
- <http://maintainable.com>
- <http://ohloh.net/accounts/mnaberez>

# Introduction

# Routes

- Answers “how do I map URLs to my code?”
- Ben Bangert re-implemented the routing system from Ruby on Rails to Python.
- The Python version has now been ported to PHP 5 as Horde/Routes.

# Routes Ecosystem

Module ActionController::Routing

In: vendor/rails/actionpack/lib/action\_controller/routing/builder.rb  
vendor/rails/actionpack/lib/action\_controller/routing/optimisations.rb  
vendor/rails/actionpack/lib/action\_controller/routing/recognition\_optimisation.rb  
vendor/rails/actionpack/lib/action\_controller/routing/routes.rb  
vendor/rails/actionpack/lib/action\_controller/routing/route\_set.rb  
vendor/rails/actionpack/lib/action\_controller/routing/segments.rb  
vendor/rails/actionpack/lib/action\_controller/routing.rb

## Routing

The routing module provides URL rewriting in native Ruby. It's a way to redirect incoming requests to controllers and actions. This replaces `mod_rewrite` rules. Best of all, **Rails' Routing** works with any web server. Routes are defined in `config/routes.rb`. Consider the following route, installed by **Rails** when you generate your application:

```
map.connect ':controller/:action/:id'
```

This route states that it expects requests to consist of a `:controller` followed by an `:action` that in turn is fed some `:id`. Suppose you get an incoming request for `/blog/edit/22`, you'll end up with:

```
params = { :controller => 'blog',  
          :action   => 'edit',  
          :id       => '22'  
}
```

Think of creating routes as drawing a map for your requests. The map tells them where to go based on some predefined pattern:

```
ActionController::Routing::Routes.draw do |map|  
  Pattern 1 tells some request to go to one place  
  Pattern 2 tell them to go to another  
  ...  
end
```

The following symbols are special:

```
:controller maps to your controller name  
:action     maps to an action with your controllers
```

Other names simply map to a parameter as in the case of `:id`.

## Route priority

Not all routes are created equally. Routes have priority defined by the order of appearance of the routes in the `config/routes.rb` file. The priority goes from top to bottom. The last route in that file is at the lowest priority and will be applied last. If no route matches, 404 is returned.

Within blocks, the empty pattern is at the highest priority. In practice this works out nicely:

```
ActionController::Routing::Routes.draw do |map|  
  map.with_options :controller => 'blog' do |blog|  
    blog.show '', :action => 'list'  
  end  
  map.connect ':controller/:action/:view'
```

## Routes

Routes is a Python re-implementation of the [Rails routes system](#) for mapping URL's to Controllers/Actions and generating URL's. Routes makes it easy to create pretty and concise URL's that are RESTful with little effort.

Speedy and dynamic URL generation means you get a URL with minimal cruft (no big dangling query args). Shortcut features like Named Routes cut down on repetitive typing.

Current features:

- Named Routes
- Sophisticated Route lookup and URL generation
- Wildcard path's before and after static parts
- Groupings syntax to allow flexible URL's to accommodate almost any need
- Sub-domain support built-in
- Conditional matching based on domain, cookies, HTTP method (RESTful), and more
- Easily extensible utilizing custom condition functions and route generation functions
- Extensive unit tests

Buzzword Compliance: *REST, DRY*

## News

**Feb. 26th, 2008**

Routes 1.7.2:

- Fixed bug with keyword args not being coerced to raw string properly.

**Nov. 16th, 2007**

Routes 1.7.1:

- Fixed bug with sub-domains from route defaults getting encoded to unicode resulting in a unicode route which then caused `url_for` to throw an exception.
- Removed duplicate assignment in `map.resource`. Patch by Mike Naberezny.
- Applied test patch fix for path checking. Thanks Mike Naberezny.

## Horde/Routes

Home Install Integrate Manual

### Home

Horde/Routes tackles an interesting problem that comes up frequently in web development: how do you map a URL to your code?

There are many solutions to this problem ranging from using the URL paths as an object publishing hierarchy to regular expression matching. Horde/Routes goes a slightly different way.

Using Horde/Routes, you specify parts of the URL path and how to match them to your Controllers and Actions. The specific web framework you're using may actually call them by slightly different names, but for the sake of consistency we will use these names.

Horde/Routes lets you have multiple ways to get to the same Controller and Action, and uses an intelligent lookup mechanism to try and guarantee you the URL with the least cruft\* when generating the URL.

URL Cruft  
Shorthand reference to what will occur if a Route can't handle all the arguments we want to send it. Those arguments become HTTP query args (`/something?query=arg&another=arg`), which we try to avoid when generating a URL.

### Python Version

Horde/Routes, and even this website, are directly derived from the Routes project for Python, by Ben Bangert.

Horde/Routes is part of the [Horde Project](#).

Latest Version  
Version 0.3.0

Developers  
Mike Naberezny  
Chuck Hagenbuch

License  
Horde/Routes is free software, covered by a standard [BSD license](#) and is © 2007-2008 [The Horde Project](#).

Ruby



Python



PHP

# Routes

- Provides solutions for both recognizing URLs and generating URLs
- Standalone component that is easy to integrate and web framework agnostic
- Developed by Maintainable & Horde

# Installation

```
$ pear channel-discover pear.horde.org  
$ pear install horde/routes
```

- PEAR install
- <http://pear.horde.org/index.php?package=Routes>

# Installation

```
function sample_autoloader($class) {  
    require str_replace('_', '/', $class) . '.php';  
}  
  
spl_autoload_register('sample_autoloader');
```

- Horde/Routes uses a PEAR-like class and file naming scheme. All files use autoloading.
- Assuming you have Horde/Routes in your `include_path`, registering an autoload function like above will be all you need.



# Terminology

- A web application exposed by Routes is organized at the top-level into *controllers*
- Each *controller* is typically responsible for a single application resource (usually a noun)
  - `PostsController`
  - `CommentsController`
  - `AuthorsController`

# Terminology

- Each *controller* responds to *actions* (usually a verb) that act on a resource
- PostsController
  - `index, show, update, delete*`

# Terminology

- The *action* of a *controller* may receive other pieces of the URL as *parameters*.
- `/:controller/:action/:id`
- `/posts/show/5`

# Setting up the Mapper

# Mapper

- `Mapper` is the core of the Routes system. You `connect()` routes to the mapper.
- You can then `match()` a URL against the set of routes you have connected.

# Mapper

- As far as Routes is concerned, the list of controller names is just a list of names.
- Routes just performs matching. It's up to you or your framework to dispatch what it matches into your application structure.

# Mapper

```
$map = new Horde_Routes_Mapper();  
$map->connect('/:controller/:action/:id');  
  
print_r(  
    $map->match('/blogs/show/1')  
);  
#=> NULL
```

No match!

# Mapper

- Internally, Routes uses regular expressions to match connected routes against URLs.
- These regular expressions must be generated before routes can be matched.



# Create RegExps

- You need to `createRegs ()` on the Mapper before its routes can be matched.
- Controllers are special.
- Routes needs to know the name of every controller in your application to `createRegs ()`.

# Create RegExps

```
$map = new Horde_Routes_Mapper();  
$map->connect('/:controller/:action/:id');  
$map->createRegs(array('blogs'));  
  
print_r(  
    $map->match('/blogs/show/1')  
);  
  
#=> array('controller'=>'blogs', 'action'=>'show', 'id'=>1)
```

## Option 1

Pass a list of all controller names to `createRegs()`

# Create RegExps

```
function my_scanner($directory) { return array('blogs'); }

$map = new Horde_Routes_Mapper(array('controllerScan'=>'my_scanner',
                                     'directory'=>'/'));

$map->connect('/:controller/:action/:id');
$map->createRegs();

print_r(
    $map->match('/blogs/show/1')
);
#=> array('controller'=>'blogs', 'action'=>'show', 'id'=>1)
```

## Option 2

`controllerScan` callback builds controller list

# Create RegExps

```
# touch ./controllers/blogs.php

$map = new Horde_Routes_Mapper(array('directory'=>'./controllers'));
$map->connect('/:controller/:action/:id');
$map->createRegs();

print_r(
    $map->match('/blogs/show/1')
);
#=> array('controller'=>'blogs', 'action'=>'show', 'id'=>1)
```

## Option 3

Default `Horde_Routes_Utils::controller_scan`

# Tips

```
# touch ./controllers/blogs_controller.php

$map = new Horde_Routes_Mapper(array('directory'=>'./controllers'));
$map->connect('/:controller/:action/:id');
$map->createRegs();

print_r(
    $map->match('/blogs/show/1')
);
#=> array('controller'=>'blogs', 'action'=>'show', 'id'=>1)
```

- `alwaysScan` will cause `createRegs()` called before any `match()`.
- This is useful mostly during development.

# Tips

```
$map = new Horde_Routes_Mapper(array('directory'=>'./controllers'));  
  
print_r(  
    call_user_func($map->controllerScan, $map->directory)  
);  
#=> array('blogs')
```

- Call `controllerScan` for sanity if routes don't match when you think they should.

# Review

- Create Mapper Instance
- Connect Routes to the Mapper
- Generate Regular Expressions
- Match or Generate

# Route Recognition



# Path Parts

# Path Parts: Static

```
$map = new Horde_Routes_Mapper();
$map->connect('atom',
    array('controller'=>'feeds', 'action'=>'show', 'format'=>'atom'));
$map->connect('rss2',
    array('controller'=>'feeds', 'action'=>'show', 'format'=>'rss2'));
$map->createRegs(array('feeds'));

print_r( $map->match('/atom') );
#=> array('controller'=>'feeds', 'action'=>'show', 'format'=>'atom')

print_r( $map->match('/rss2') );
#=> array('controller'=>'feeds', 'action'=>'show', 'format'=>'rss2')
```

- Both routes have static paths: `atom` and `rss2`

# Path Parts: Dynamic

```
$map = new Horde_Routes_Mapper();
$map->connect('feeds/:format',
             array('controller'=>'feeds', 'action'=>'show'));
$map->createRegs(array('feeds'));

print_r( $map->match('/feeds/atom') );
#=> array('controller'=>'feeds', 'action'=>'show', 'format'=>'atom')

print_r( $map->match('/feeds/rss2') );
#=> array('controller'=>'feeds', 'action'=>'show', 'format'=>'rss2')
```

- Static part: `feeds`
- Dynamic part: `:format`

# Path Parts: Wildcard

```
$map = new Horde_Routes_Mapper();  
$map->connect('folders/:action/*folder_path',  
             array('controller'=>'folders'));  
$map->createRegs(array('folders'));  
  
print_r( $map->match('/folders/show/path/to/somewhere') );  
#=> array('controller'=>'folders', 'action'=>'show',  
         'folder_path'=>'path/to/somewhere')
```

- Static part: `folders`
- Dynamic part: `:action`
- Wildcard part: `*folder_path`

# Defaults

# Defaults

```
$map = new Horde_Routes_Mapper();
$map->connect(':title',
             array('controller'=>'posts', 'action'=>'show'));
$map->createRegs(array('posts'));

print_r(
    $map->match('/all-about-routes')
);
#=> array('controller'=>'posts', 'action'=>'show',
         'title'=>'all-about-routes')
```

- Routes are free-form. Controller and action do not need to be part of the URL itself.

# Implicit Defaults

```
$map = new Horde_Routes_Mapper();  
$map->connect(':title');  
$map->createRegs(array('posts'));  
  
print_r(  
    $map->match('/all-about-routes')  
);  
#=> array('controller'=>'content', 'action'=>'index',  
          'title'=>'all-about-routes')
```

- Gotcha. Notice magic `content` and `index`
- `Mapper (explicit=False)` is standard, giving all routes implicit defaults

# Defaults

```
$map = new Horde_Routes_Mapper();
$map->connect('archives/:year', array('controller'=>'posts',
                                     'action'=>'show_archive', 'year'=>'2008'));
$map->createRegs(array('posts'));

print_r( $map->match('/archives') );
#=> array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'2008')

print_r( $map->match('/archives/2005') );
#=> array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'2005')
```

- Defaults are used to implement optional parts of the URL (`year`)



# Requirements & Conditions

# Requirements

```
$map = new Horde_Routes_Mapper();  
$map->connect('archives/:year', array('controller'=>'posts',  
                                     'action'=>'show_archive', 'year'=>'2008'));  
$map->createRegs(array('posts'));  
  
print_r( $map->match('/archives/2005') );  
#=> array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'2005')  
  
print_r( $map->match('/archives/rat') );  
#=> array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'rat')
```

- “Year of the `rat`” is probably not something that we want to support.

# Requirements

```
$map = new Horde_Routes_Mapper();
$map->connect('archives/:year',
    array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'2008',
        'requirements'=>array('year'=>'\d{4}')));
$map->createRegs(array('posts'));

print_r( $map->match('/archives/2005') );
#=> array('controller'=>'posts', 'action'=>'show_archive', 'year'=>'2005')

print_r( $map->match('/archives/rat') );
#=> NULL
```

- Requirements help cut down on validation in application code. Be specific.

# Conditions

```
$map = new Horde_Routes_Mapper();
$map->connect('posts/create',
    array('controller'=>'posts', 'action'=>'create',
        'conditions'=>array('method'=>array('POST'))));
$map->createRegs(array('posts'));

$map->environ = array('REQUEST_METHOD'=>'POST');
print_r( $map->match('/posts/create') );
#=> array('controller'=>'posts', 'action'=>'create')

$map->environ = array('REQUEST_METHOD'=>'GET');
print_r( $map->match('/archives/rat') );
#=> NULL
```

- Routes can enforce conditions on the request environment in addition to requirements on the URL itself.

# URL Generation

# URL Generation

```
$map = new Horde_Routes_Mapper();  
$map->connect('/:controller/:action/:id');  
$map->createRegs(array('articles'));  
  
$utils = $map->utils;  
print_r(  
    $utils->urlFor(array('controller'=>'articles',  
                        'action'=>'show', 'id'=>3))  
);  
#=> /articles/show/3
```

- Generating URLs allows the structures to change without changing the application code

# Named Routes

```
$map = new Horde_Routes_Mapper();
$map->connect('home', 'articles',
             array('controller'=>'articles', 'action'=>'index'));
$map->createRegs(array('articles'));

$utils = $map->utils;
print_r(
    $utils->urlFor('home')
);
#=> /articles/
```

- We can give a name to each route as we connect them. This should be considered a best practice and makes generation easier.

# More

- Static Named Routes
- Filter Functions
- Grouping Path Parts
- More conditions:  
subdomain, function
- RESTful Routes
- Mapper .routematch()
- Redirects



# Resources

- Narrative documentation  
<http://dev.horde.org/routes/>
- Issue tracking  
<http://bugs.horde.org>
- CVS repository  
<http://horde.org/source/>  
<http://cvs.horde.org/framework/Routes/>

# Q & A

# Thanks!