URL Mapping with Horde/Routes

php|works 2008
Mike Naberezny

http://maintainable.com

About Me

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

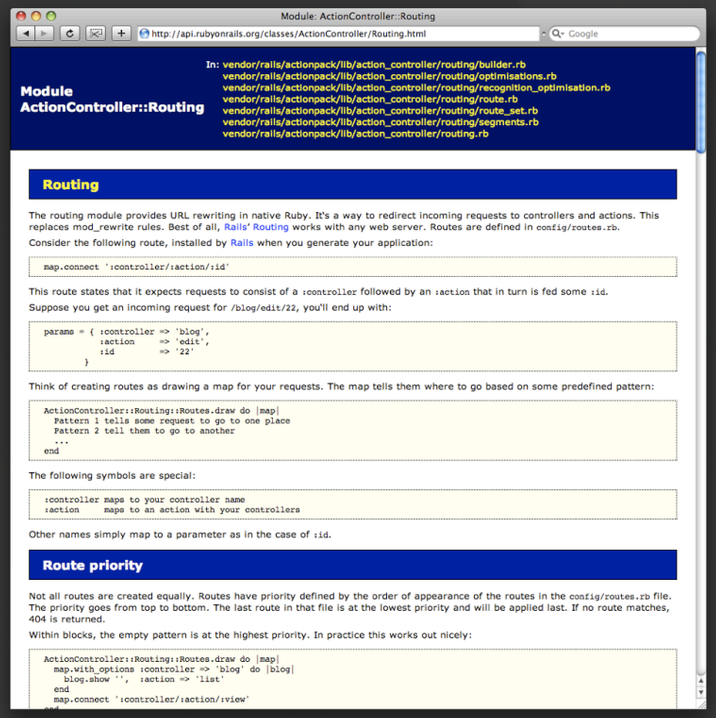
Maintainable Software

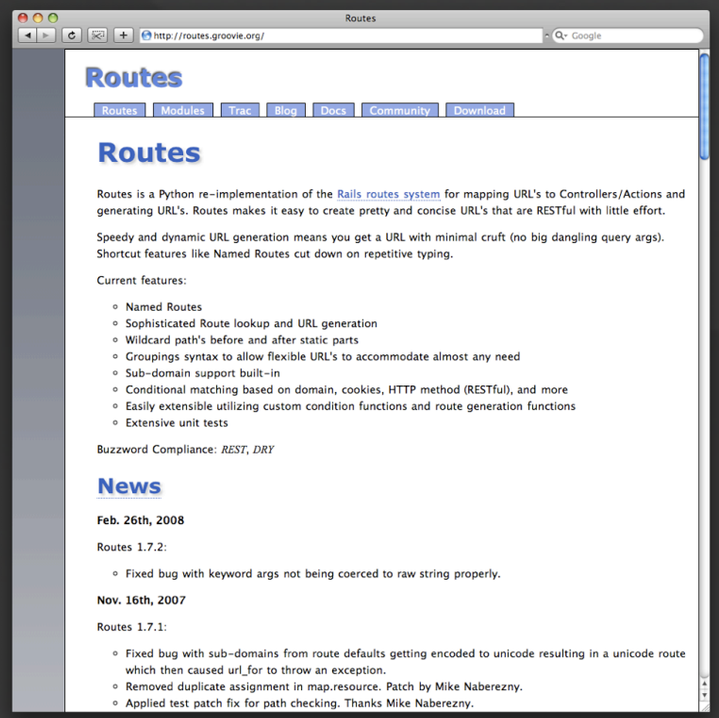
Introduction

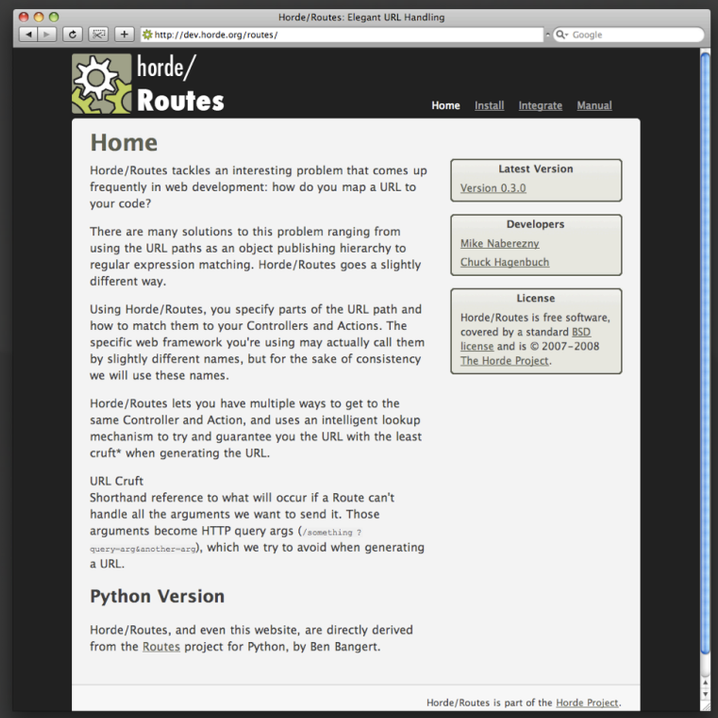
Routes

- 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







Ruby → Python ↔ PHP

Maintainable Software

http://dev.horde.org/routes

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

Maintainable Software

Installation

```php
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`*

**Rails calls this "destroy"*

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

```php
$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

```php
$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

```php
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

* *directory is optional*

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

```php
$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

```php
$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`

Maintainable Software

Path Parts: Dynamic

```php
$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`

Maintainable Software

Path Parts: Wildcard

```php
$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

```php
$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.

Maintainable Software

Implicit Defaults

```php
$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

Maintainable Software

Defaults

```php
$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`)

Maintainable Software

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.

Maintainable Software

Requirements

```php
$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.

Maintainable Software

Conditions

```php
$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.

Maintainable Software

URL Generation

URL Generation

```php
$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

```php
$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

Maintainable Software

Thanks!