



Minimalistic PHP Framework for Backends

User Guide v. 0.6

Overview

Rapyd is minimalistic php framework made to build applications based on two patterns:

- MVC (Model, View, Controller) flow pattern
- CRUD (Create, Read, Update, Delete) data pattern.

Furthermore, It provides a set of components / classes that can help you to develop web applications "rapidly".

Its syntax is simple, oriented to build simple (or complex) Data GUI in few line of code.

It's inspired by CodeIgniter and KohanaPHP both great (and probably better) frameworks not focused on data editing which is the "plus" of Rapyd.

I must also to mention CaffeinePHP, which is a valid alternative to rapyd if you are on PHP4 hosting.

Installation

System Requirements

1. PHP 5.1 or higher (needed)
2. Use of .htaccess (apache with mod_rewrite) (apache needed, mod_rewrite optional)
3. DB wrapper currently support (mysql, pdo-sqlite3) (db is optional)

Configuration & Deployment

You can choose the SVN version (you can also ask for a developer branch) or a standard release (download from rapyd.com unzip and enjoy).

You must configure application in /application/config.php (locale, path, db connection). Then you're ready to start to develop your application.

Rapyd comes with a Demo module (with several examples of data presentations and editing) and a Welcome controller. You can remove both.

Understand the application flow is enough to begin to develop.

URLs and Flow

Rapyd URLs

Rapyd uses URI (uniform resource identifiers) to determine which area and which action of your application to load/call.

So instead classic “php application” urls:

`http://website/welcome.php`
`http://website/register.php?step=1`

It use:

`http://website/index.php/welcome/`
`http://website/index.php/registration/submit`

Or (with a simple .htaccess you can find in the zip the index.php can be omitted)

`http://website/welcome/`
`http://website/registration/submit`

In the above urls rapyd will find:

- a controller: “welcome”, then it will call the “index” method (because the second segment is empty and the default action is “index”).
- A controller: “registration”, then it will call the “submit” method (because the second segment IS the requested action).

In the next pages we will see what is a “controller”.
Understand how to build a controller is enough to begin to develop.

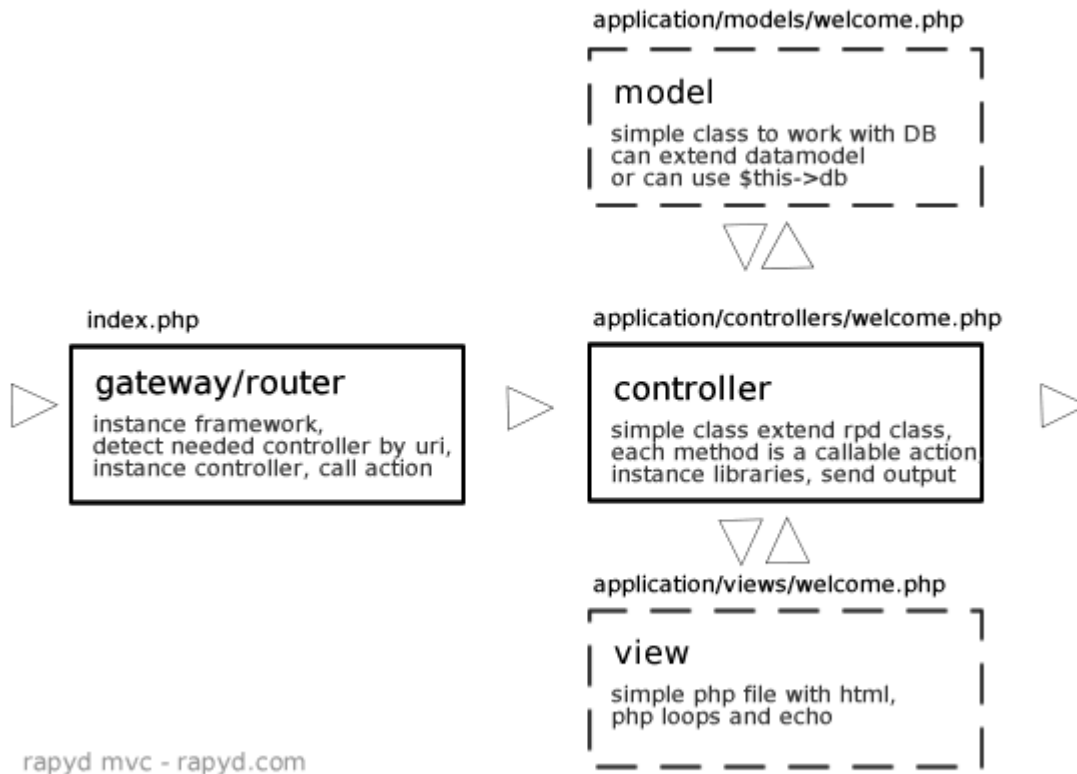
Application Flow

request

<http://website/index.php/welcome>

response

html output



rapyd mvc - rapyd.com

In Rapyd, and other MVC frameworks:

- All non static resources are served by a single gateway (index.php).
- The gateway detect by uri the needed resource and instance the right "Controller".
- A Controller is a simple class where each Method is a callable action.
- A Controller Method can instance libraries, perform DB queries, send an output (echo..)
- Optionally a Controller Method can use a "model" to work with DB and a "view" to format output response.
- A Model is a simple class where you can isolate db stuffs.
- A View is a php file with html output, php loops, echos etc..

As you can see in the picture, the flow is:

request > **gateway** > **controller** [> **model**] [> **view**] > **output**

Rapyd also support "modules" which are like "sub-applications", and cascade resource load, so you can isolate pieces, and can extend each single system library.

Filesystem

▼	application	
▷	assets	static files (images, css, js etc..)
▷	controllers	controllers (welcome.php, register.php, ecc..)
▷	libraries	custom classes, you can also extend system libraries
▷	models	custom db classes (myauth.php, user.php, ecc..)
▷	views	views (welcome.php, register_success.php, ecc..)
	config.php	configuration file
▷	modules	modules folder (each module has the same filesystem struct. of application)
▼	rapyd	system folder (probably you don't need to touch here)

Rapyd Filesystem is clean, “system” is separated from “application” (usually you can upgrade by replacing “rapyd” folder).

Furthermore, you can isolate application pieces in “modules” folder (so you can organize a cms in modules).

If you need to customize “rapyd/libraries” you can build your extended version in “application/libraries”

Autoload: you do not have to worry about load classes before instancing or extending, Rapyd autoload will find and include all needed files (even if they are in a different folder/module).

Controller

A controller is a simple php class it must extend rpd superclass (so it can inherit `$this->db`, `$this->url` and some other system libraries).

Conventions/rules:

- Class name must end with `_controller`
- Filename must be minuscapse, with the same class name (without `_controller` suffix)
- File must be deployed in `/application/controllers/` (or in `/modules/modulename/controllers/`)

Sample

```
<?php
class welcome_controller extends rpd {

    function index()
    {
        $vars = array('varname' => 'value');
        echo $this->view('welcome', $vars);
    }

    function test()
    {
        echo 'test';
    }
}
```

This request:

http://website/welcome

will parse the View 'welcome' then the result will be printed

This other request:

http://website/welcome/test

will output 'test'

View

A view is a simple php file with html, php loops, echos etc...

Conventions/rules:






- Filename must be minuscapse
- File must be deployed in /application/views/ (or in /modules/modulename/views/)

Sample

```
<html>
<head>
  <title>Welcome !</title>
</head>
<body>
  <h1>Welcome</h1>
  <p>This is a simple html file with some php var <?=$varname?> </p>
</body>
</html>
```

To see how you can send vars from a controller see the Controller sample.

DataGrid

Article List			
ID	△▽	Title	Body
 7		Title 7	Body 74444
 8		Title 8	<p><strong
 9		Title 9	Body 9
 10		Title 10	Body 10 da
 11		pippo	C R A P<fo
< Prima < 1 2 3 4 5 6 > Ultima >			

Sample

```
$grid = new datagrid_library();
$grid->label = 'Article List';
$grid->per_page = 5;

$grid->source('articles');
$grid->column('article_id', 'ID', true)
    ->url('edit?show={article_id}', 'detail.gif');

$grid->column('title', 'Title');
$grid->column('body', 'Body')->callback('escape');

$grid->build();

$data['head'] = $this->head();
$data['content'] = $grid->output;
```

Properties

Property	Default Value	Options	Description
label	"	string	label to display for the grid
per_page	10000'	integer	the number of records to display per page

Methods

\$grid->source(\$source)

\$source - mixed may be a db-table name, a sql-query, a datafilter object, an associative array matrix

\$grid->column(\$pattern, \$label, \$is_orderby)

\$pattern - string field pattern, field name, or content of cells
\$label - string column label
\$orderby - boolean if column is sortable

\$grid->build()

build grid, fill \$grid->output

DataFilter

Article Filter
Title
Public Yes No

Sample

```
$filter = new datafilter_library();
$filter->label = 'Article Filter';
$filter->db->select("articles.*, authors.*");
$filter->db->from("articles");
$filter->db->join("authors", "authors.author_id=articles.author_id", "LEFT");
$filter->field('input', 'title', 'Title')
  ->attributes(array('style' => 'width:170px'));
$filter->field('radiogroup', 'public', 'Public')
  ->options(array("y"=>"Yes", "n"=>"No"));

$filter->buttons('reset', 'search');
$filter->build();

$data['head'] = $this->head();
$data['content'] = $grid->output;
```

Properties

Property	Default Value	Options	Description
label	"	string	label to display for the filter

Methods

\$filter->source(\$source)

\$source - mixed may be a db-table name, or a sql-query

\$filter->field(\$type, \$name, \$label)

\$type - string field type (input, password, checkboxgroup, checkbox, radiogroup, radio, dropdown, date, editor)
\$name - string field name, usually the db table field to append in the where clause
\$label - string label to display for the field

\$filter->field(\$type, \$name, \$label)->attributes(\$attributes)

\$attributes - assoc. array array of extra attributes to build for column

\$filter->field(\$type, \$name, \$label)->options(\$options)

\$options - array array of options (for field types like dropdown, radiogroup etc..)

\$filter->buttons(\$button1[, \$button2...])

\$button n - mixed name of buttons to build ('reset' and 'search' are the standard buttons available for datafilter)

\$filter->build()

build filter, fill \$filter->output

DataEdit

Manage Article Modifica

Title	Title 2
-------	---------

Public	Yes
--------	-----

Author	Jhon
--------	------

Date	01/04/2009
------	------------

Description

Body 2!! kjkjkjkjыыык

Torna all'elenco

Sample

```
$edit = new dataedit_library();
$edit->label = 'Manage Article';
$edit->back_url = $this->url('filtered_grid');

$edit->source('articles');
$edit->field('input','title','Title')->rule('trim','required');

$edit->field('radiogroup','public','Public')
->options(array("y"=>"Yes", "n"=>"No"));

$edit->field('dropdown','author_id','Author')
->options('SELECT author_id, firstname FROM authors')
->rule('required');

$edit->field('date','datefield','Date')
->attributes(array('style'=>'width: 80px'));

$edit->field('editor','body','Description')->rule('required');

$edit->buttons('modify','save','undo','back');

$edit->build();

$data['head'] = $this->head();
$data['content'] = $edit->output;
```

Properties

Property	Default Value	Options	Description
label	"	string	label to display for the filter
back_url	"	string	url of page where to go back (when we click on back button)

Methods

\$edit->source(\$source)

\$source	-	mixed	may be a "datamodel" object (or extended one), or the name of a db table (in this case dataedit will instance a new datamodel for us)
-----------------	---	-------	---

\$edit->field(\$type, \$name, \$label)

\$type	-	string	field type (input, password, checkboxgroup, checkbox, radiogroup, radio, dropdown, date, editor)
\$name	-	string	field name, usually the db table field to append in the where clause
\$label	-	string	label to display for the field

\$edit->field(\$type, \$name, \$label)->attributes(\$attributes)

\$attributes	-	assoc. array	array of extra attributes to build for column
---------------------	---	-----------------	---

\$edit->field(\$type, \$name, \$label)->rule(\$rule)

\$rule	-	mixed	\$rule can be 'required' or can be the name of a custom or native php function (like trim), it's possible to pass an array instead a single rule, or use a serialized syntax like 'required trim' using a pipe as separator
---------------	---	-------	---

\$edit->field(\$type, \$name, \$label)->options(\$options)

\$options	-	array	array of options (for field types like dropdown, radiogroup etc..)
------------------	---	-------	--

\$edit->buttons(\$button1[, \$button2...])

\$button n	-	mixed	name of buttons to build ('modify', 'save', 'undo' and 'back' are the standard buttons available for dataedit)
-------------------	---	-------	--

\$edit->build()

build dataedit, fill \$edit->output