|  |  |
| --- | --- |
|  | Kilabite Tech Ltd.  Information Technology |

PHPForms

User Manual

Version 1.0

15/11/2017

Table of Contents

[1. Introduction 3](#_Toc26906195)

[2. Detailed Description 4](#_Toc26906196)

[2.1 Installation 4](#_Toc26906197)

[2.2 Architecture 4](#_Toc26906198)

[2.3 Inbuilt Functionality 7](#_Toc26906199)

[2.3.1 kbFormsLog 7](#_Toc26906200)

[2.3.2 kbFormsRoute 7](#_Toc26906201)

[2.3.3 kbSetSystemFolder 7](#_Toc26906202)

[2.3.4 kbFormsGetRoute 7](#_Toc26906203)

[2.3.5 Template and View Variables 8](#_Toc26906204)

[3. Views 10](#_Toc26906205)

[3.1 Functionality 10](#_Toc26906206)

[3.1.1 kbFormsView 10](#_Toc26906207)

[3.1.2 kbTemplateView 11](#_Toc26906208)

# Introduction

PHPForms is a development tool that integrates directly within PHP to facilitate development using the MVC development paradigm.

**Why PHPForms?**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Easy- to- Use**  PHPForms provides a simple and intuitive development interface. |  | **Reduce Development Time**  Manage your time and create templates for fast solution delivery |
|  | **Debugging**  Easier debugging features within PHP |  | **RAD**  Web Rapid Application Development |
|  |  |  |  |

Kilabite Tech Ltd.

kevin@kilabite.com

# Detailed Description

PHPForms is an extension that makes PHP operate in an MVC model automatically. It also has some simple helper functions that should make a PHP developer’s life easy. Currently only PHP 5.6 is supported and on Linux CentOS and Ubuntu. Please do contact us is you require a different build at kevin@kilabite.com

## Installation

Copy the file kilabite.so to the PHP extensions folder. On CentOS that would probably be /usr/lib64/php/modules/ however if you do a phpinfo(); you will get the correct installation path.

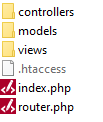
In your php.ini add the following line:

kilabite.basefolder=/var/www/html/

i.e. the root path for your websites

## Architecture

The core php command is **kbFormsRoute**



**.htaccess** (optional) user defined rules care to be taken to rewrite the url correctly if that’s used

**Index.php** contains one line

<?php

kbFormsRoute("Home/Index/0");

?>

**router.php**

*File Structure*:

<user global definitions>

<user global preparations>

<must have instructions as follows>

(Replace **mysystem** with your system path)

include\_once "controllers/Controller.inc";

foreach (glob("controllers/\*.php") as $filename)

{

include $filename;

}

kbSetSystemFolder(isset($\_SESSION["user"]) ? $\_SESSION["user"] : "", "**mysystem**/","Home/Index/1");

if (isset($\_GET['method']) && isset($\_GET['class']))

{

$strMethod = $\_GET['method'];

$\_class = $\_GET['class'];

$object = new $\_class();

if(is\_callable(array($object, $strMethod))){

isset($\_GET['param']) ? $object->$strMethod($\_GET['param']) : $object->$strMethod();

}else{

die(json\_encode(array('Error'=>true,'msg'=>'router: bad mvc call')));

}

}

*Full Example*:

<?php

define("ITEM\_LABEL\_POS", 0);

define("DB\_IP\_ADDRESS", "localhost");

define("DB\_USERNAME", "mytest");

define("DB\_PASSWORD", "testpassword");

define("DB\_DBNAME", "mytest");

define("DB\_PORT", "3306");

date\_default\_timezone\_set('Europe/Rome');

error\_reporting(-1);

if (!isset($GLOBALS['conn'])) {

$GLOBALS['conn'] = @mysqli\_connect(DB\_IP\_ADDRESS\_PORT, DB\_USERNAME, DB\_PASSWORD, DB\_DBNAME);

if (mysqli\_connect\_errno()) {

die('Could not connect: ' . mysqli\_connect\_error());

}

}

ini\_set('memory\_limit', '-1');

ini\_set('max\_execution\_time', 600);

include\_once "controllers/Controller.inc";

foreach (glob("controllers/\*.php") as $filename)

{

include $filename;

}

kbSetSystemFolder(isset($\_SESSION["user"]) ? $\_SESSION["user"] : "", "**mysystem**/","Home/Index/1");

if (isset($\_GET['method']) && isset($\_GET['class']))

{

$strMethod = $\_GET['method'];

$\_class = $\_GET['class'];

$object = new $\_class();

if(is\_callable(array($object, $strMethod))){

isset($\_GET['param']) ? $object->$strMethod($\_GET['param']) : $object->$strMethod();

}else{

die(json\_encode(array('Error'=>true,'msg'=>'router: bad mvc call')));

}

}

?>

In the views folder there must be an index.html. No jquery is required however to use the template view ctl functionality the following are needed

Jquery

Bootstrap

SelectPicker

DatetimePicker

## Inbuilt Functionality

### kbFormsLog

**Parameters**: string

string, number

Outputs the value of variables to the PHP error log as a warning

*Example:*

$t = “Hello”;

kbFormsLog($t)

*Output*:

PHP Warning: Hello

### kbFormsRoute

**Parameters**: string

Given an MVC path this function builds a full URL and redirects to that location

kbFormsRoute(<route path>);

*Example:*

$t = “Hello”;

kbFormsRoute(“Home/show/”.$t);

### kbSetSystemFolder

**Parameters**: string, string, string

kbSetSystemFolder(<username>,<filename>,<route path>);

Used mainly in the main php router file but where ever else developer may want to change system routing

*Example:*

kbSetSystemFolder(isset($\_SESSION["user"]) ? $\_SESSION["user"] : "", "**mysystem**/","Home/Index/1");

### kbFormsGetRoute

**Parameter**: string

Given an MVC path this function builds a full URL and returns it as a string

kbFormsGetRoute(<route path>);

*Example:*

$t = “Hello”;

echo kbFormsGetRoute (“Home/show/”.$t);

*Output:*

http://mysystem.example/HomeController&param=Hello

### Template and View Variables

@{<variable>}

PHPForms takes the variable name and looks it up in the model that was passed. If its not found then PHPForms checks to see if it’s a php function. Should one be found that is executed.

Example:

*PHP*

$model = array(“YesNo”=>“<option value=’YES’>Yes</option><option value=’NO’>No</option>”);

kbFormsView(“mytest”,$model);

*HTML*

<select>@{ YesNo}</select>

This tag would be replaced as follows

<select><option value=’YES’>Yes</option><option value=’NO’>No</option></select>

**Note that functions to be called from views must always have at least one parameter. This is the filename which would be past for debugging purposes**

*PHP*

function getYesNo($file)

{

return “<option value=’YES’>Yes</option><option value=’NO’>No</option>”;

}

<select>@{ getYesNo}</select>

This tag would be replaced as follows

<select><option value=’YES’>Yes</option><option value=’NO’>No</option></select>

More complex method calling takes a parameter. A parameter can be hardcoded or a variable

Example

*PHP*

Function getYesNo($file, $param)

{

If ($param == “Simple”)

return “<option value=’YES’>Yes</option><option value=’NO’>No</option>”;

else

return “<option value=’YES’>Yes of course!</option><option value=’NO’>No! Never!</option>”;

}

<select>@{ getYesNo :Complex}</select>

This tag would be replaced as follows

<select><option value=’YES’>Yes of course!</option><option value=’NO’>No! Never!</option></select>

Method calling can also do inline evaluation of parameters. A parameter can be hardcoded or a variable

Example

*PHP*

$model = array(“test”=>0);

kbFormsView(“mytest”,$model);

*HTML*

<select>@{ test == 1 <option>Choice 1</option><option>Choice 2</option>}</select>

This tag would be replaced as follows

<select></select>

*Valid operators for numeric and string depending on type are:*

**== < >**

**!= || &&**

**>= <=**

# Views

## Functionality

### kbFormsView

**Parameters**: string

string, array

Loads a named view and optionally passes on to it a model. The model is a set of data variables that can be used by the view. These are identified within the view by the following syntax

@{<variable>} as described in 2.2.5

Example:

Basic call. This calls mytest.html and looks for it in the views folder

kbFormsView(“mytest”);

More complex call

*PHP*

$model = array("replace"=>"my text", "another" => "hello world");

kbFormsView(“mytest”,$model);

*mytest.html*

<script>

function pressButton()

{

$("#mybutton").html("@{another}");

}

</script>

<button id="mybutton" onclick="pressButton()">@{replace}</button>

### kbTemplateView

**Parameters**: array

array, array

Template views are different in that they automatically create for you a table with all fields as specified and if needed in shorthand. The form controls have a lot of inbuilt pre-setting setup specifically for bootstrap. Template Views are denoted by .ctl extension. They are the same as kbFormsView but take extra commands to build a form as follows. At a basic level ctl takes a grid command followed by the number of columns (n) required and optionally a width (w) value

{

<grid *n [w]*>

<…. Controls ….>

}

The following parameters can be passed in the first array:

'f': filename of ctl required

't': title of the table. optional

'x': cancel button is displayed and the value is the javascript function to call

's': save button is displayed and the value is the javascript function to call

'u': update button is displayed and the value is the javascript function to call

'd': delete button is displayed and the value is the javascript function to call

'c': number of columns wanted for the form. Overrides hardcoded value in ctl

'p': padding wanted for the form

'w': width size wanted for the form. Overrides hardcoded value in ctl

In all cases a meaningful word can be used as long as it starts with the correct instruction letter

Example

*PHP*

public function Personal($param)

{

$\_m = new Employees();

$model = $\_m->getPersonalInfo($param);

$model['room']='not assigned';

$model['viewonly'] = "";

$\_function = "updatePersonalInfo()";

if (intVal($param) == 0) {

$model['readonly'] = "";

$\_function = "insertPersonalInfo()";

} else {

$model['readonly'] = "readonly='readonly'";

}

}

kbTemplateView(array("filename"=>"employees/personal","save"=>$\_function,"col"=>"6","padding"=>"10"), $model);

*CTL*

{

<grid 4>

<b onclick="uploadPicture()" class='btn-outline-warning' @{disable}><k class='icon-user icon-white'><l UPLOAD PHOTO>

<c Started><i type="text" required id="admission\_date" value="@{admission\_date}" @{viewonly} data-date-format="DD-MM-YYYY" name="d:admission\_date" readonly>

<c Balance owing><input id="medical\_insurance" name="medical\_insurance" value="@{medical\_insurance}" @{viewonly} type="text" placeholder="Balance owing">

<c Referral><v class="autocomplete" style="width:300px;"><input autocomplete="off" id="admitted\_from" name="admitted\_from" @{viewonly} value="@{admitted\_from}" type="text" placeholder="Community" required="required">

<c Notes><t id='medical\_history' name='medical\_history' rows="4" cols="50" placeholder='NOTES' @{readonly}><f @{medical\_history}>

}

Control commands:

N.B. control commands uses classes defined in bootstrap however if a class directive is found the class is added to the base bootstrap classes. So there is no need to add form-control and btn class

*LABEL*  'l'

Example:

<label This is my label text>

OR

<l This is my label text>

*INPUT*  'i'

<input type="text" required id="admission\_date" value="@{admission\_date}" @{viewonly} data-date-format="DD-MM-YYYY" name="d:admission\_date" readonly>

OR

<i type="text" required id="admission\_date" value="@{admission\_date}" @{viewonly} data-date-format="DD-MM-YYYY" name="d:admission\_date" readonly>

**N.B. In case of dates the name has to be preceded by d: this tells PHPForms to use the standard bootstrap date input**

<i type="number" id="mynumber" name="mynumber">

*CTRL\_LABEL* 'c'

Creates a label for a control. A control must follow

<c Balance owing><input id="medical\_insurance" name="medical\_insurance" value="@{medical\_insurance}" @{viewonly} type="text" placeholder="Balance owing">

*DIV* 'v'

Builds a div. Must be followed by a control example label

Example

<v ><l my text label>

*BUTTON* 'b'

Builds a button. Must always be followed by a label tag even if empty

<b onclick="uploadPicture()" class='btn-outline-warning'>

*TEXTAREA* 't'

Builds a textarea. Must always be followed by a freetext tag even if empty

<t id='medical\_history' name='medical\_history' rows="4" cols="50" placeholder='NOTES' @{readonly}><f @{medical\_history}>

*FREETEXT* 'f'

<freetext This is all just some free text>

*IKON* 'k'

Builds an icon. Must always be followed by a label tag even if empty

<k class='icon-user icon-white'><l UPLOAD PHOTO>

*CALENDAR*  'd'

Builds a bootstrap datepicker calendar input. This is the same as having a normal input as described above

<c Started><d id='d:dob' autocomplete='off' data-date-format='DD-MM-YYYY' onblur='calcAge()' name='d:dob' value='@{dob}' placeholder='DD-MM-YYYY' @{viewonly}>

*REQUIRED* 'r'

Outputs the red star for a required field. Must be followed by a control

<c Referral><r ><input autocomplete="off" id="admitted\_from" name="admitted\_from" @{viewonly} value="@{admitted\_from}" type="text" placeholder="Community" required="required">

*SELECT*  's' and *OPTION* 'o'

Creates a select dropdown. Must be followed by at least one option tag

<c Medical Insurance><s id="medical\_insurance" name="medical\_insurance" ><o Yes><o No>

Basic call. This calls myctl.ctl and looks for it in the views folder

*php*

kbTemplateView(array("filename"=>"myctl","save"=>”saveData”,"col"=>"6","padding"=>"10"));

More complex call

*PHP*

$model = array("replace"=>"my text", "another" => "hello world");

kbTemplateView(array("filename"=>"myctl","save"=>”saveData”,"col"=>"6","padding"=>"10"), $model);

*mytest.ctl*

<script>

function pressButton()

{

$("#mybutton").html("@{another}");

}

</script>

<button id="mybutton" onclick="pressButton()">@{replace}</button>

<div class='container'>

<form id="the-admission-form" class='form' method='' action=''>

<div class='card'>

<div class='card-header'>

<div class='card-header'>

<h4 class='card-title'>Admission</h4>

</div>

</div>

<div class='card-body'>

<input name='id' value='@{id}' type='hidden'/>

<input name='number' value='@{number}' type='hidden'/>

<input name='user' type='hidden'/>

<!-- input name='picture\_path' value='views/residents/photos/resident@{p\_id}.png' type='hidden'/ -->

<img id="camera" style="width:340px" src='views/residents/photos/resident@{p\_id}.jpg'/>

{

<grid 4>

<b onclick="uploadPicture()" class='btn-outline-warning' @{disable}><k class='icon-user icon-white'><l UPLOAD PHOTO>

<c Admission Date><r ><i type="text" required id="admission\_date" value="@{admission\_date}" @{viewonly} data-date-format="DD-MM-YYYY" name="d:admission\_date" readonly>

<c Medical History><input id="medical\_history" name="medical\_history" value="@{medical\_history}" @{viewonly} type="text" placeholder="Rheumatoid Arthritis (RA)">

<c Admitted From><r ><v class="autocomplete" style="width:300px;"><input autocomplete="off" id="admitted\_from" name="admitted\_from" @{viewonly} value="@{admitted\_from}" type="text" placeholder="Community" required="required">

<c Medical Insurance><s id="medical\_insurance" @{disable} name="medical\_insurance" >@{getMedicalInsurance :medical\_insurance}

<c Active Care Plan> <input readonly name="active\_care\_plan" id="active\_care\_plan" placeholder="Care Plan 1" type="text" value="@{active\_care\_plan}" >

<c Active Care Plan Reassesed> <select @{disable} name="active\_care\_plan\_reassesed" id="active\_care\_plan\_reassesed" placeholder="Care Plan 1" > @{getYesNo :active\_care\_plan\_reassesed}

<c Number of Checks Per Night> <input name="number\_of\_checks\_per\_night" id="number\_of\_checks\_per\_night" placeholder="Number of Checks Per Night" type="number" value="@{number\_of\_checks\_per\_night}" >

<c Comorbidities> <s id="comorbidities" @{disable} name="comorbidities" class="selectpicker" data-live-search="true" multiple><o >

<input id="comorbidities\_other" name="comorbidities\_other" @{disable} readonly='readonly' data-role="tagsinput" class="tagsinput badge-azure badge-fill">

}

</div>

</div>

</form>

</div>