Database Project
Form helper and Validation(code-igniter)

User Management Group

16/05/2012

What is code-igniter?

- Is a lightweight web application framework written in PHP that adopts the model-view-controller approach to development.
- Why use code-igniter?
  - Feature rich
  - Light weight
  - Open source
  - Excellent “by example” Documentation
  - Easy to configure
  - Support multiple databases

Helper Functions

- Helpers, as the name suggests, help you with tasks.
- Each helper file is simply a collection of functions in a particular category. There are URL Helpers, that assist in creating links, there are Form Helpers that help you create form elements, Text Helpers perform various text formatting routines, Cookie Helpers set and read cookies, File Helpers help you deal with files, etc.
- We will discuss Form Helper here.

What is Form Helper?

- The Form helper provides methods to assist you in creating forms.
- Loading this Helper
  - Loading code

```
$this->load->helper('form');
```

form_open()

- Creates an opening form tag with a base URL built from your configure preferences.
- Here's a simple example:
  - `echo form_open('email/send');`
- The above example would create a form that points to your base URL plus the "email/send" URI segments, like this:
  - `<form method="post" accept-charset="utf-8" action="http:/example.com/index.php/email/send" />`

Adding Attributes

- Attributes can be added by passing an associative array to the second parameter, like this:
  - `$attributes = array('class' => 'email','id' => 'myform');`
  - `echo form_open('email/send', $attributes);`

- The above example would create a form similar to this:

```
<form method="post" accept charset="utf8" action="http:/example.com/index.php/email/send"  class="email"  id="myform" />
```

form_hidden()

- It generates hidden input fields. You can either submit a name/value string to create one field:
  - form_hidden('username', 'johndoe');

  // Would produce:
  - <input type="hidden" name="username" value="johndoe" />
- Or you can submit an associative array to create multiple fields:
  - $data = array('name' => 'John Doe',
    'email' => 'john@example.com',
    'url'   => 'http://example.com');
  - echo form_hidden($data);

  // Would produce:
  - <input type="hidden" name="name" value="John Doe" />
  - <input type="hidden"name="email"value="john@example.com" />
  - <input type="hidden" name="url" value="http://example.com" />

form-input ()

- It generate a standard text input field. You can minimally pass the field name and value in the first and second parameter:
  - echo form_input('username', 'johndoe');
- Or you can pass an associative array containing any data you wish your form to contain:
  - $data = array('name' => 'username',
    'id' => 'username'
    'value' => 'johndoe',
    'maxlength' => '100',
    'size' => '50',
    'style' => 'width:50%',
    );
  - echo form_input($data);

Form input ()

// Would produce:

- <input type="text" name="username" id="username" value="johndoe" maxlength="100" size="50" style="width:50%" />
- All others identical functions:
  - form_password()
  - form_upload()
  - form_textarea()
  - form_dropdown() : Create a standard drop-down field. The first parameter will contain the name of the field, the second parameter will contain an associative array of options, and the third parameter will contain the value you wish to be selected. You can also pass an array of multiple items through the third parameter, and CodeIgniter will create a multiple select for you.
  - form_multiselect()

form_checkbox()

Lets you generate a checkbox field. Simple example:

- echo form_checkbox('newsletter', 'accept', TRUE);

// Would produce:

<input type="checkbox" name="newsletter" value="accept" checked="checked" />

- The third parameter contains a boolean TRUE/FALSE to determine whether the box should be checked or not.

form_checkbox()

```
$data = array(
    'name'        => 'newsletter',
    'id'          => 'newsletter',
    'value'       => 'accept',
    'checked'     => TRUE,
    'style'       => 'margin:10px',
    );
echo form_checkbox($data);
// Would produce:
<input type="checkbox" name="newsletter" id="newsletter" value="accept" checked="checked" style="margin:10px" />
```

- Form_radio is identical in all respects to the form_checkbox().

form_submit()

- Generate a standard submit button. Simple example:

```
echo form_submit('mysubmit', 'Submit Post!');

// Would produce:

<input type="submit" name="mysubmit" value="Submit Post!" />
```

- Similar to other functions, you can submit an associative array in the first parameter if you prefer to set your own attributes. The third parameter lets you add extra data to your form, like JavaScript.

form_label()

- Generate a <label>. Simple example:

```
echo form_label('What is your Name', 'username');
// Would produce:
<label for="username">What is your Name</label>
```

- to set additional attributes.

```
$attributes = array(
  'class' => 'mycustomclass',
  'style' => 'color: #000;',                    );
echo form_label('What is your Name', 'username', $attributes);
// Would produce:
<label for="username" class="mycustomclass" style="color:
#000;">What is your Name</label>
```

- form_reset() identical to form label()

form_button()

```
echo form_button('name','content');

// Would produce

<button name="name" type="button">Content</button>

$data = array(
    'name' => 'button',
    'id' => 'button',
    'value' => 'true',
    'type' => 'reset',
    'content' => 'Reset'                );

echo form_button($data);

// Would produce:

<button name="button" id="button"
value="true"type="reset">Reset</button>
```

form_close()

- Produces a closing </form> tag. The only advantage to using this function is it permits you to pass data to it which will be added below the tag. For example:

```
$string = "</div></div>";

echo form_close($string);

// Would produce:

</form>

</div></div>
```

Form validation

- Form validation is CI library for validating forms to use this library we have to load this library in Controller

  $this->load->library("form_validation")

- if we want to use form validation we need 3 files

  1. controller
  2. view where form exists
  3. a view file where we have a success message.

To set rule

- In controller file we can check the form and can set the rules for each form elements.
- $this->form_validation->set_rules("nameOfFormElement", "Purpuse of that Element", "Rules")

  Example

  $this->form_validation->set_rules('userName', 'User Name', 'required');

  we can set as many rules as we want.
- Rules: required, matches, is_unique, min_length[ ], max_length[ ], exact_length, numeric, integer, decimal, valid_email, valid_ip, and so on.

To set rule

- If we want to use multiple validation simply we use | operator between each rule.

  example: $this->form_validation->set_rules('userName', 'User Name', 'required|min_length[5]|max_length[12]|is_unique[users.username]');

  we can also set rules as array of rules and than pass to set_rules().

  example: $rules = array(

```
                    array('field' => 'userName',
                          'label' => 'User Name',
                          'rules' => 'required|min_length[5]') ,
                   array('field' => 'password',
                          'label' => 'Password',
                   'rules' => 'required|min_length[6]|alpha_numeric') )
$this->form_validation->set_rules($rules);
```

To set rule

- After setting rules, when form submited we simply call the following function

```
if($this->form_validation->run() == False) {
$this->load->view("myform");    //agian load the same form
}
else {
$this->load->view("formsuccess");
// go to another page and show success message
}
}
```

- How to show error messages when form not submitted correctly?

```
echo validation_errors();
```

how to repopulate the forms fields when form_validation failed

- simply we set the value of each field with

  (set_value('nameOfField')) function

  example :

  <input type="text" name="username" value="<?php echo set_value('username'); ?>" >

More functions...

- we can call a call_back function during validation.
- we can change the error messages with
  set_message('rule', 'Error message');

  where rule is a specific rule and Error message is the message we want,
- we can also display error message as we want

  by default errors displays as

  but we can set this delimiter by own

  example :

  $this->form_validation->set_error_delimiters(' ', ' ');

More Functions...

- Or when we display error messages we can set the delimiters

  validation_errors(' ', ' ');

- we can show each error individually

  form_error('fieldName');

More Functions...

- As we had set_value('fieldName'); function to re-populate the form we have such a function for combobox (select) e.g set_select('selectName', 'value')
- For checkbox we have set_chekbox('checkbox[ ]', 'value');

  and for radio buttons we have set_radio('radioName', 'value')
- We can also prep our data using validation rules

  example: $this->form_validation->set_rules('username', 'User name', 'required|min_length[6]|xss_clean);

  or : $this->form_validatoin->set_rules('password', 'Password', 'required|min_length[5]|md5);
- The xss_clean method clean the text from malicious code, and md5 encrypt the password.

Thank you!