Zend_Form - Simon Mundy & Ralf Eggert

<ac:macro ac:name="info"><ac:parameter ac:name="title">Proposal has been revised in-depth on 14 October 2006. Please take a look at revision 0.2 thoroughly if you have already read and commented revision 0.1</ac:parameter></ac:macro>

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Form Component Proposal

Proposed Component Name	Zend_Form
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Form
Proposers	Simon Mundy Ralf Eggert
Revision	0.1 - 28 August 2006: Initial proposal. 0.2 - 14 October 2006: revised proposal. (wiki revision: 16)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Form model
Validation and Filtering
Extended form fields
View helpers
Automatic rendering
Automatic form generation
Form processor
Form controller
6. Milestones / Tasks
7. Class Index
Form model
View helpers
Automatic rendering
Automatic form generation
Form processor
8. Use Cases
UC 01: Use the form model with your own home made validation methods
UC 02: Template view that renders the form using view helpers
UC 03: Automatic rendering
UC 04: Use an array with all form values and fields to be used by Smarty
UC 05: Define the form model
UC 06: Use the form model to validate and render a form using Zend_View
UC 07: Handling error messages
UC 08: Automatic form generation
Use Case #9: Form processor
9. Class Skeletons

1. Overview

Zend_Form is a component for creating, validating and processing form data.

2. References

- PEAR_QuickForm
- Zend_View form helpers
- patForms
- HTML_QuickForm2
- Complete XForms Examples
- XForms for HTML Authors

3. Component Requirements, Constraints, and Acceptance Criteria

- One or multiple instances per page
- Provides an interface for creating forms for a variety of end-uses (XHTML for current proposal, other end-uses to be implemented)
- Allows auto or manual data population (can utilise Zend_Filter_Input)
- Allows fine-grain control of form optional parts for display, validation, filtering and generation

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Input_Filter
- Zend_Filter (tbc)
- Zend_Validator (tbc)

5. Theory of Operation

Zend_Form provides a set of loosely coupled parts and view helpers to model, validate & filter, display, generate and process a form. The core of Zend_Form is the form model which is the only mandatory part. All other parts are optional, yet all parts work together smoothly with the form model.

Form model

With the form model the programmer will be able to model a specific form. It is an abstract class which needs to be extended to implement a specific form model. The model class provides the required functionalities to:

- add the unique id for the form
- add form fields
- set defaults for each form field
- accept values for the form fields (e.g. from raw $_POST data, a Zend_Filter_Input object or even a simple array)
- pass all form attributes, values and fields to an array

It also provides an abstract setup method which needs to be implemented in an extended form model class to add fields to the specific form.

Every form field is an object with a name property, an unset Zend_Validator property, an unset Zend_Filter object and a simple isValid() method.

Validation and Filtering

Since the validation and filtering part is optional the programmer can use his own validation method. Anyway the validation and filtering part provides the required functionalities to:

- add a Zend_Validator object with validation rules (e.g. Zend_Rule_Range) to a field of the form
- add a Zend_Filter object with filter rules (e.g. Zend_Filter_Int) to a field of the form
- add allowed keys for a selection using a Zend_Rule_IsIn validation rule
- validate the form
- get the error status of invalid form fields

Based on the added Zend_Validator objects a form renderer will get some relevant information about how to display a specific form field, e.g. to display the "maxlength" attribute for text input fields or to specify if a select takes only one or more options. Also the allowed select options will be set with validation rules (Zend_Rule_IsIn).

Extended form fields

There are a set of form field objects which extend the basic Zend_Form_Field. They do not add much more functionality to Zend_Form_Field except setting up the Zend_Validator object for the form field with a set of validation rules and setting up the Zend_Filter object with a set of filter rules.

For example the Zend_Form_Field_Text sets up the Zend_Validator object with the Zend_Rule_String validation rule. Optional it takes a range for the text field and adds a Zend_Rule_Range validation rule to the Zend_Validator. Selections which expect only one option to be selected, can be handled by Zend_Form_Field_Enum. Selections which expect none, one or more options to be selected can be handled by Zend_Form_Field_Set.

Zend_Form_Field can easily be extended by the programmer to add custom validation and filter rules.

View helpers

The view helpers are optional as well and extend the currently available view helpers by taking a form object to generate the output (e.g. XHTML) for

- text fields
- password fields
- textareas
- checkboxes
- radio buttons
- select lists
- file upload fields

All view helpers take a reference to an field of the form object to identify values and other attributes set by the form object. The view helpers also examine the attached Zend_Validator object for the output (e.g. to set the "maxlength" for text inputs). Furthermore the view helpers take a list of other attributes for displaying the form field (e.g. "class", "size", "rows", "title").

For a select list, checkboxes and radio buttons the view helper will add the keys and corresponding values to display. Enum selections can be display as single selects or radio buttons. Set selections can be displayed as multiple selects or checkboxes.

To display validation errors there is a view helper to get an array of all unmatched validation rules for a specific form field or the whole form. Based on this the appropiate errors messages can be displayed.

Note: it still needs to be discussed if the current view helpers (release 0.1.5) should be kept or overwritten.

Automatic rendering

Again the rendering module is optional, so the programmer can handle the rendering of the form in every way he wants to, for example with the view helpers mentioned above. The rendering module will provide the required functionalities to

- take further display attributes for each form field
- organize form fields into sections (fieldsets)
- generate the output code (e.g. XHTML) for a single field
- generate the output code (e.g. XHTML) for the complete form

Note: the layout of these generated forms will mostly be limited to a linear layout but can be changed with CSS.

Automatic form generation

This optional module generates a form model with all fields from a record of a table (Zend_Db_Table_Row) or from a XML data record. Also some validation rules will automatically be added when appropriate, e.g. set maxixum chars, allowed keys for selections, etc.).

The generated form class can be cached or saved and extended with additional validation and filter rules.

Form processor

This optional module accepts validated form data and processes it according to business needs (**Note**: Darby, please specify in more detail, what a form processor should do and how it should do it).

Form controller

This optional part extends the Zend_Controller_Action to add a set of methods to handle form building, form validation, form rendering and form processing and can be extended to build specific controllers.

6. Milestones / Tasks

- Milestone 1: Form building and form rendering with Zend_View helpers (UC 01, UC 02)
- Milestone 2: Render form completely and render form to an array (UC 03, UC 04)
- Milestone 3: Form validation and filtering with Zend_Validator and Zend_Filter (UC 05, UC 06, UC 07); NOTE: can not be completed until Zend_Validator and Zend_Filter can be used!
- Milestone 4: Automatic form generation (based on a database table or an xml file) and form processing (UC 08, UC 09)
- Milestone 5: Improve the client side of the form handling with Javascript validation, AJAX interaction and more.
- Milestone 6: Handle Xforms

7. Class Index

Form model

- Zend_Form
- Zend_Form_Exception (extends Zend_Exception)
- Zend_Form_Field
    - Zend_Form_Field_Text
    - Zend_Form_Field_Enum
    - Zend_Form_Field_Set
    - Zend_Form_Field_File

View helpers

- Zend_View_Helper_FormElement (acts as a factory to a Zend_Form_Render object)

Automatic rendering

- Zend_Form_Renderer_Abstract

- Zend_Form_Renderer_Xhtml
    - Zend_Form_Renderer_Xhtml_Input
    - Zend_Form_Renderer_Xhtml_Text
    - Zend_Form_Renderer_Xhtml_Select
    - Zend_Form_Renderer_Xhtml_Textarea
    - Zend_Form_Renderer_Xhtml_Checkbox
    - Zend_Form_Renderer_Xhtml_Radio
    - Zend_Form_Renderer_Xhtml_Button
    - Zend_Form_Renderer_Xhtml_Hidden
    - Zend_Form_Renderer_Xhtml_File
- Zend_Form_Renderer_Xform

Automatic form generation

- Zend_Form_Generator_Abstract
    - Zend_Form_Generator_Db
    - Zend_Form_Generator_Xml

Form processor

- Zend_Form_Processor

8. Use Cases

The use cases will build a simple form with the following form elements:

- 'username': an input type=text for username (minlength = 5, maxlength = 32)
- 'password': an input type=password for password (minlength = 5, maxlength = 16)
- 'email': an input type=text for email
- 'language': a single select for a language selector with an associative array ('en' => 'English', 'oz' => 'Strine', 'nz' => 'Kiwi');
- 'accept': a checkbox for site acceptance

9. Class Skeletons

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>