

Zend_Form

<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	Matthew Weier O'Phinney
Revision	0.9.0 - 19 December 2007: Initial proposal creation 0.9.1 - 03 January 2008: Minor formatting and grouping changes and corrections 0.9.2 - 03 January 2008: Updates to address community feedback 0.9.3 - 04 January 2008: Add grouping to proposal 0.9.4 - 05 January 2008: Added information on validator/filter enhancements, additional view helpers 0.9.5 - 08 January 2008: Added pluginLoader details to class skeletons 0.9.6 - 09 January 2008: Updated to reflect decorator usage 0.9.7 - 10 January 2008: Updated to reflect current API 1.0.0 - 17 January 2008: Updated with link to initial docs (wiki revision: 26)

Table of Contents

1. Overview
 2. References
 3. Component Requirements, Constraints, and Acceptance Criteria
 4. Dependencies on Other Framework Components
 5. Theory of Operation
- Overview
- Zend_Form_Element
- Typical workflow:
- Standard elements to ship with first iteration:
- Standard decorators to ship with first iteration:
- Zend_Form
6. Milestones / Tasks
 7. Class Index
 8. Use Cases
- Creating form programmatically
- Create custom form class
- Validating partial forms
- Validating and returning JSON response
- Render a form
- Render an individual element
- Group elements
- Decorators
9. Class Skeletons

1. Overview

NOTE: Initial documentation is now available

Form processing is a routine and common task for web developers that involves many facets: creation of form HTML, creation of server side validation, and escaping form elements for display on-screen or insertion into data storage; in today's web, we also must consider whether or not a user will be able to interact dynamically with a form element using AJAX.

Considering that web forms are the key to dynamic content on websites, usage of forms should be as easy as possible. Form validation and filtering logic should be encapsulated; if possible, auto-generation of HTML would be desirable.

Zend_Form proposes to do the following things:

- Form element objects encapsulate the following:
 - Validation chains
 - Filter chains
 - Rendering
 - Decorator classes for elements and forms
 - Default decorator classes for HTML generation
 - Allow multiple decorators
 - Localization hinting (through [Zend_View_Helper_Translate](#))
 - Ability to set element state from a config file
- Form object encapsulates all form elements
 - Setting current locale in all elements
 - single entry point for validating all elements at once
 - accessors for retrieving individual form elements
 - ability to generate full HTML
 - loops over all form elements and renders them
 - methodology for evaluating AJAX-submitted forms
 - allows validating only passed parameters, not entire form
 - Ability to create all form elements and set form state from a config file

2. References

- [Simon Mundy's Zend_Form proposal](#)
- [Jurriën Stutterheim's Zend_Form proposal](#)
- [Mitchell Hashimoto's Zend_Form proposal](#)
- [Ruby on Rails](#)
- [Symfony project](#)
- [Solar_Form](#)
- [Django](#)
- [CGI::Application::Plugin::ValidateRM](#)
- [AjaxContext action helper](#)
- [JSON view helper](#)

3. Component Requirements, Constraints, and Acceptance Criteria

- **Must** be de-coupled from Zend_Controller and Zend_View
 - Form setup and validation **must** be possible without Zend_Controller
 - **Should** accept data to validate, and not retrieve it automatically from the environment
 - View rendering **should** be **optional**, and allow for usage of **either** Zend_View or custom rendering functionality
- **Must** provide separation between validation, filtering, and rendering
 - **Must** be flexible enough to allow non-XHTML rendering schemas
 - **Must** have a concept of required/non-required elements
 - **Must** have methods for:
 - Validation of entire form
 - Validation of partial form
 - Hinting that a request was done via AJAX
- **Must** re-use existing components as much as possible
 - [Zend_Filter](#)
 - [Zend_Json](#)
 - [Zend_Locale](#)
 - [Zend_Translate](#)
 - [Zend_Validate](#)
 - [Zend_View_Helper_*](#)
- **Must** be able to do both automated and manual rendering of form items
 - Automated rendering:
 - **Must** be capable of displaying labels, form elements, and error messages
 - **Must** re-use existing form helpers when possible
 - **Must** be capable of localization

- **Must** be AJAX-friendly
 - **Should** allow validation of single elements or groups of elements, with a single response
 - Response **should** default to JSON serialization in a documented format
 - Response format **should** be configurable via subclassing
 - **Should** define element IDs
 - **Should** allow defining arbitrary HTML accessors for use with JS (on* attributes and/or JS-library specific)
 - **Could** provide a div for error message display
- **Must** be able to order form elements
 - Default **should** be order in which elements are added to form object
- **Must** be able to load form and individual form elements from config objects
- **Could** be capable of handling multi-page forms (i.e., saves form state between requests)
 - **Should** provide examples of how multi-page forms could be handled even if no implicit handling is provided.
- **Should** allow grouping of form elements for display and/or validation purposes
- **Could** allow lazy-loading of validators and/or filters when needed
- **Should** either implement validator/filter factories, or require their development
- **Could** be extended to allow direct interaction with a model class (such as a Zend_Db_Table)
- **Must** provide examples of common Ajax patterns
 - Autocompletion
 - Validation of single/multi elements over AJAX

4. Dependencies on Other Framework Components

- Zend_Config (optional)
- Zend_Exception
- Zend_Filter_*
- Zend_Json (optional; only used with AJAX)
- Zend_Locale (optional, through Zend_View_Helper_Translate)
- Zend_Loader_PluginLoader (for use with factories)
- Zend_Translate (optional, through Zend_View_Helper_Translate)
- Zend_Validate_*
- Zend_View (optional - for rendering and localization)

5. Theory of Operation

Overview

Zend_Form encapsulates any number of Zend_Form_Elements. Each element contains its own validation and filter chains, as well as mechanisms for rendering (be it via view helpers or other mechanisms).

Zend_Form contains metadata about the form itself, including potentially any HTML attributes used to define the form element. It then has methods for validating against all elements or a subset of elements, retrieving errors and error messages (if any), retrieving individual elements, and rendering the form.

Zend_Form_Element will contain metadata about individual elements. Overloading will be provided to allow setting arbitrary metadata, and by default used as HTML element attributes. Additionally, each element will contain its own validator and filter chains, and hinting about how to render itself.

Elements and the form object itself will each have decorators associated with them. By default, elements will use 'ViewHelper', 'Label', 'Errors', and 'HtmlTag' decorators, which will allow using existing Zend_View helpers for rendering the element, label, and errors, and surround the element in an HTML div. Zend_Form_ElementGroup will surround its content using 'ViewHelper' to select the Fieldset view helper, and Zend_Form will use a new Form view helper. Alternate decorators may be specified at any time, and may be nested (first in will be innermost) or chained (with options to prepend or append to content).

Zend_Form and Zend_Form_Element will each have accessors for setting a Zend_Translate object, allowing localization of the form. The various Zend_View_Helper_Form* view helpers will be retrofitted to allow translation, if a translation object is present (most likely by utilizing the proposed Zend_View_Helper_Translate view helper).

Developers will be able to specify Filters and Validators using strings that indicate the full class name and/or the short class name, as well as

constructor options. This will enable using `Zend_Config` to configure a form object. Additionally, validators used with `Zend_Form` will be passed a second argument to `isValid()`, `$context`, which will contain all elements being validated; this will allow validating elements in relation to other submitted values, if required. Finally, all filters and validators will be accessible by name, allowing modification in as well as removal from their respective chains.

Separate proposals for generic AJAX integration with the Zend Framework MVC will be created, including:

- [AjaxContext](#) action helper, for switching view context when XHR requests are detected and based on the response format requested
- [JSON view helper](#), for returning JSON responses
- [Autocompleter action helper](#), for serializing an array to JSON and returning a response

Zend_Form_Element

Functionality includes:

- `Zend_Validate` validator chain; attach as few or as many validators as needed
 - Allow specifying validators as:
 - objects
 - full class names
 - short class names (minus prefix)
- `Zend_Filter` filter chain; attach as few or as many filters as needed
 - Allow specifying filters as:
 - objects
 - full class names
 - short class names (minus prefix)
- Allows marking element as required/optional (allowing skipping validation if empty or not present)
- `getValue()` retrieves filtered value by default
- `getRawValue()` for retrieving original provided value
- Accessor for setting label
- Accessors for retrieving errors and messages
- Accessors for setting arbitrary attributes (possibly via overloading)
- Optionally locale aware
 - if so, all messages and labels will be localized. Configures `Zend_View_Helper_Translate` with `Zend_Translate` object provided to element.
- Method for validating
- Methods for manipulating decorators for rendering an element
- Method for rendering
 - Uses any attached decorators
 - `__toString()` will proxy to `render()`
- Method for loading state via a `Zend_Config`

Typical workflow:

```
$username = new Zend_Form_Element_Text('username');
$username->setLabel('User name')
    ->addValidator(new Zend_Validate_NotEmpty())
    ->addValidator('EmailAddress')
    ->addFilter('StringToLower');
if ($username->isValid()) {
    echo "Welcome, ", $username, "!"; // "Welcome, foobar@foo.com!"
} else {
    $messages = $username->getMessages();
    echo "Error in validation:\n    ", implode("\n    ", $messages);
}
```

Standard elements to ship with first iteration:

- Button
- Checkbox
- Hidden
- Image
- Multiselect
- Password

- Radio
- Reset
- Select
- Submit
- Textarea
- Text

Standard decorators to ship with first iteration:

- ViewHelper

Zend_Form

Functionality includes:

- Accessors for adding, removing, and retrieving Zend_Form_Element objects
 - Adding elements allows optionally setting order
- Accessors for setting arbitrary attributes (possibly via overloading)
- Iterable; iterates over attached elements
- Accessor for setting locale
- getValues() retrieves filtered element values
- getValue(\$name) retrieves single filtered element value
- getRawValues() retrieves unfiltered element values
- getRawValue(\$name) retrieves single unfiltered element value
- Accessors for retrieving errors and messages for all values or single value
- Methods for validating:
 - full form (isValid())
 - partial form (isValidPartial())
 - AJAX versions of each of the above
 - Returns standardized JSON response
- Methods for manipulating decorators for rendering the form
- Method for rendering
 - Uses associated decorators
 - __toString() proxies to it
- Method for loading state of form and all elements via a Zend_Config
- (optional) Ability to group elements into sections/pages

6. Milestones / Tasks

- Milestone 1: [DONE] Create prototype
- Milestone 2: [DONE] Create proposal and submit for community review
- Milestone 3: [DONE] Finalize form element base code and individual elements, including tests
- Milestone 4: [DONE] Finalize form base code, including tests
- Milestone 5: [DONE] Test rendering, including JSON responses
- Milestone 6: Documentation, demos, and tutorials
- Milestone 7: Component moved to core

If a milestone is already done, begin the description with "[DONE]", like this:

- Milestone #: [DONE] Unit tests ...

7. Class Index

- Zend_Form classes:
 - Zend_Form
 - Zend_Form_Decorator_Interface
 - Zend_Form_Decorator_Abstract
 - Zend_Form_Decorator_Errors
 - Zend_Form_Decorator_HtmlTag
 - Zend_Form_Decorator_Label
 - Zend_Form_Decorator_ViewHelper
 - Zend_Form_Decorator_ViewScript
 - Zend_Form_Element

- Zend_Form_ElementGroup
- Zend_Form_Element_Exception
- Zend_Form_Element_Autocomplete
- Zend_Form_Element_Button
- Zend_Form_Element_Checkbox
- Zend_Form_Element_Hidden
- Zend_Form_Element_Image
- Zend_Form_Element_Multi
- Zend_Form_Element_Multiselect
- Zend_Form_Element_Password
- Zend_Form_Element_Radio
- Zend_Form_Element_Reset
- Zend_Form_Element_Select
- Zend_Form_Element_Submit
- Zend_Form_Element_Textarea
- Zend_Form_Element_Text
- Zend_Form_Element_Xhtml
- Zend_Form_Exception
- Zend_View helpers:
 - Zend_View_Helper_Fieldset
 - Zend_View_Helper_Form
 - Zend_View_Helper_FormErrors

8. Use Cases

9. Class Skeletons

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