

Zend_Form - Jurriën Stutterheim

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



Archived

This proposal has been archived, and deprecated in favor of the consolidated [Zend_Form Proposal](#).

<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	Jurriën Stutterheim
Revision	1.3 - 30 September 2007: Revised. (wiki revision: 21)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
 - Zend_Form_Factory
 - Zend_Form
 - Zend_Form_Element
 - Zend_Form_Storage
 - View helpers
 - Action helper
 - Plugins
 - Builders
 - Zend_Form_Filter_Input
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Form is a component that manages form filtering/validation and, form data/status across multiple pages.

2. References

- [Zend_Form](#) - Simon Mundy & Ralf Eggert
- [QuickForm2](#)
- My own library with Form implementation

3. Component Requirements, Constraints, and Acceptance Criteria

- **Must** provide separation between form building and form handling.
- **Must** provide view helpers to easily render form elements or even an entire form.
- **Must** provide action helpers to automate form handling
- **Must** be able to handle data from multiple pages
- **Must** provide a way to store a multi-page form in multiple storage containers (Session, Cache)
- **Could** utilize `Zend_Filter_Input` for form validation and filtering
- **Must not** be bound to one output format. Developers must be able to use the `Zend_Form` for other output than just XHTML.

4. Dependencies on Other Framework Components

- `Zend_Cache_Core*`
- `Zend_Controller_Action_Helper_Abstract`
- `Zend_Exception`
- `Zend_Filter_*`
- `Zend_Filter_Input`
- `Zend_Session`
- `Zend_Validate_*`
- `Zend_View`

5. Theory of Operation

The basic usage is as follows:

A form page is instantiated. The form processes the submitted data (if available).

If not available, pass the form page to the view and render the form.

If available, validate the form and forward to the appropriate action.

To prevent expired forms there should always be a redirection after the form has been handled.

Zend_Form_Factory

The `Zend_Form_Factory` class is a factory which makes it easy to generate a `Zend_Form`. It also makes sure the right datasource is used to populate the form and that the form is serialized in storage.

Zend_Form

The `Zend_Form` represents one form as displayed on a screen. The form contains a collection of fields (called elements, for cross-format naming) and can be validated as a whole.

The entire form can be serialized into a storage container, so it may be reused in another request. It stores the result of a validation attempt, so the requirements for a form can be checked.

(e.g. shop wizard example where you don't want to access the payment form before user data is entered)

Zend_Form_Element

A form element contains field-specific filter and validation rules, which will be read from the form page and used for validation.

It can also contain a type-hint, so view helpers can automatically display the proper output format for the field.

There are basically two different types of form field: `Zend_Form_Element` and `Zend_Form_Element_Set` (the upload field is ignored for now).

The former can be used for textfields, textareas, hidden fields etc. The latter can be used for radiobuttons, checkboxes, select lists etc.

Zend_Form_Storage

When a form is validated, the submitted form data and the validation result will be stored for reuse. The `Zend_Form_Storage` provides a way to do this.

There are two storage methods at the moment: Session and Cache. The latter needs a pre-setup `Zend_Cache` instance.

View helpers

All `Zend_Form_Elements` of the form page are accessible in the view. The view helpers make it easy to generate XHTML for your forms. The default Zend Framework views can be used for this, but there are also a few extra view helpers for `Zend_Form`.

- The `FormField` only needs the `Zend_Form_Element` instance and decides which view helper to use based on that. In order for this to work a typehint must be set.
- The `MakeForm` helper generates an entire form by providing the form page instance. This should only be used for testing/mockup purposes for now. It may be possible to combine this with a `Layout` or `View Enhanced` implementation to make the output customizable.
- The `FormElement` helper already exists in the ZF today. It has been modified to accept a `Zend_Form_Element` as argument.
- `FormError` reads the validation errors from the form instance and displays them.
- `FormStart` generates a `<form>` tag and a hidden field that contains the unique identifier for that form.

Action helper

- The `FormHandler` action helper is a basic skeleton for the workflow described in the operations part. It also has support for form plugins, which will be discussed later.
- The `FormRedirector` is a subclass of the default `Redirector`. It contains a `formForward` and `formRedirect` method, which make sure the unique form identifier gets send.

Plugins

The plugins can be registered in the action helper. The plugins make the basic workflow customizable, so form handling can be largely automated. There are three stock plugins:

- `Location`: this redirects or forwards the controller to the specified location, based on which button was pressed to submit the form, the status of the form and optionally the status of specific form elements.
- `Modal`: checks if the form is modal and if the requirements for it are met.
- `Upload`: this handles file uploads.

Builders

The builders make it possible to generate a form instance in an easy way. The most basic builder is the `Formname` builder. As the name suggests, it takes the name of a form to make an instance. Other builders make it possible to generate a form instance from a `Zend_Config` instance or a `Zend_Db_Table_Abstract` instance.

Zend_Form_Filter_Input

As the name suggests this is a subclass of `Zend_Filter_Input`. It does not require complete filter/validator chains in the constructor and has a few methods to allow dynamic adding and removing of rules.

6. Milestones / Tasks

- Milestone 1: Make clear what to expect from a `Zend_Form` component
- Milestone 2: Define an API
- Milestone 3: Have a working prototype with examples
- Milestone 4: Unit tests
- Milestone 5: Complete documentation
- Milestone 6: Complete the component

7. Class Index

- Zend_Form
- Zend_Form_Builder...
 - _Abstract
 - _Config
 - _Db...
 - _Abstract
 - _Db2
 - _Mysqli
 - _Oracle
 - _Pdo...
 - _Mssql
 - _Mysql
 - _Oci
 - _Pgsq
 - _Sqlite
 - _Exception
 - _Form
 - _Formname
- Zend_Form_Element
 - Zend_Form_Element_Set
- Zend_Form_Element_Hint...
 - _HTML
 - _PDF
 - _XForms
- Zend_Form_Exception
- Zend_Form_Factory
- Zend_Form_Plugin...
 - _Abstract
 - _Broker
 - _Location
 - _Modal
 - _Upload
- Zend_Form_Storage
 - Zend_Form_Storage_Cache
 - Zend_Form_Storage_Abstract
 - Zend_Form_Storage_Session
- Zend_View_Helper...
 - _FormElement
 - _FormError
 - _FormField
 - _FormStart
 - _MakeForm
- Zend_Controller_Action_Helper...
 - _FormHandler
 - _FormRedirector

8. Use Cases

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

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