Zend_Filter_Input redesign - Bill Karwin

<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_Filter_Input Component Proposal

Proposed Component Name	Zend_Filter_Input
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Filter_Input
Proposers	Bill Karwin Darby Felton
Revision	1.1 - 10 April 2007: initial writeup. (wiki revision: 18)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

This is a proposed solution to apply multiple Zend_Filter and Zend_Validate actions to multiple inputs, e.g. $_GET or $_POST.

2. References

- Perl's HTML::FormValidator
- Zend_Validate error messages proposal - Bill Karwin
- Zend_Validate_Builder, Zend_Filter_Builder proposal - Bryce Lohr

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** create an object interface to filtering and validating.
- This component **will** accept an array of inputs in the form of an associative array, e.g. $_GET or $_POST superglobals.
- This component **will** allow a developer to specify a single filter or validator in a scalar.
- This component **will** allow a developer to specify multiple filters and multiple validators to apply to a given input field.
- This component **will** load, instantiate, and invoke the Filter or Validate objects.
- This component **will** allow a developer to declare additional constraints, such as required fields.
- This component **will** return filtered and validated field values in escaped format using accessors.

- This component **will** have a magic `_get()` accessor that returns the value in an escaped format, after it has been filtered and validated.
- This component **will** support user-defined Filter and Validate classes in namespaces other than Zend_Filter and Zend_Validate.

4. Dependencies on Other Framework Components

- Zend_Filter_Interface
- Zend_Filter_Exception
- Zend_Filter (filter chain class)
- Zend_Validate_Interface
- Zend_Validate_Exception
- Zend_Validate (validator chain class)
- other concrete classes that implement Zend_Filter_Interface and Zend_Validate_Interface

5. Theory of Operation

Creating an instance of Zend_Filter_Input involves declaring an array of filters and validators to apply to data fields by name. This associative array maps from a field name to a filter (or validator), or a chain of filters (or validators). In the example below, the field 'month' will be filtered by Zend_Filter_Digits and then by Zend_Filter_StringTrim.

```
$filters = array(
    'month' => 'digits'
);
```

The key of the array above is the name of the field to which to apply the filters. The value can be a scalar if one filter is desired, or an array if a chain of multiple filters is desired. Each value can be a string, which is mapped to a class name, or else an instance of an object that implements Zend_Filter_Interface. In the example below, the field 'month' will be filtered by Zend_Filter_Digits and then by Zend_Filter_StringTrim.

```
$filters = array(
    'month' => array('digits', new Zend_Filter_StringTrim())
);
```

Integer-indexed elements of the value array correspond to validators. String-indexed elements of the value array specify **metacommands**. For instance, if the key of the `$filters` array is not the same as the name of the field, you can specify it:

```
$filters = array(
    'month' = array(
        'digits',       // filter name at integer index [0]
        'field' => 'mo' // field name at string index ['field']
    )
);
```

The `$validators` array is similar to the `$filters` array. The validator has an additional metacommand called 'presence'. If its value is 'required' then if the field is not present it is reported as a **missing** field. Fields that are not declared in the validator array at all but appear in the input are reported as an **unknown** field. Fields that don't pass their validation are reported as an **invalid** field.

To create an object of Zend_Filter_Input, pass array arguments for the `$filters` and `$validators` declarations.

```
$input = new Zend_Filter_Input($filters, $validators);
```

You can add data either as the third argument to the constructor, or with the `setData()` method.

```
// Add data in constructor
$input = new Zend_Filter_Input($filters, $validators, $data);

// Replace the data, but not the declared filters and validators
$intput->setData($newData);
```

If you have user-defined filter or validator classes that don't exist in the Zend_Filter or Zend_Validate namespace, you can add more namespaces in the ctor options or with the `addNamespace()` method. You cannot remove Zend_Filter and Zend_Validate as namespaces, you can only add namespaces. User-defined namespaces are searched first, Zend namespaces are searched last.

```
$options = array('namespace' => 'My_Namespace');
$input = new Zend_Filter_Input($filters, $validators, $data, $options);

$input->addNamespace('Other_Namespace');

// Now the search order is: My_Namespace, Other_Namespace, Zend_Filter, Zend_Validate
```

Filters are applied before validators. Don't declare filters intended for escaping output in the `$filters` array. It could make the validators' job awkward. There is an opportunity to add a filter to escape output that runs after the validators.

After the filters and validators are done, you can get reports of missing, unknown, and invalid fields

```
if ($input->hasInvalid()) {
  $list = $input->getInvalid();
}

if ($input->hasMissing()) {
  $list = $input->getMissing();
}

if ($input->hasUnknown()) {
  $list = $input->getUnknown();
}
```

You can get field values in escaped format using the magic accessor. There are non-magic accessor methods for getting the field values in escaped or unescaped format.

```
$input->month;                // escaped
$input->getEscaped('month');   // escaped
$input->getUnescaped('month'); // not escaped
```

The default filter used to escape output is Zend_Filter_HtmlEntities. You can specify a different filter for escaping output. You can use the options array in the constructor, or else `setDefaultEscapeFilter()`. You can specify this filter as a string or as an object in either case.

```
// Use Zend_Filter_Trim for escaping output
$options = array('escape_filter' => 'trim');
$input = new Zend_Filter_Input($filters, $validators, $data, $options);

// Use Zend_Filter_HtmlEntities for escaping output
$input->setDefaultEscapeFilter(new Zend_Filter_HtmlEntities());
```

If you want more than one escaping filter available simultaneously in a single instance of Zend_Filter_Input, you should subclass Zend_Filter_Input and implement a new method to get values in a different escaped format.

6. Milestones / Tasks

- Milestone 1: Post prototype design, gather community feedback
- Milestone 2: Working prototype checked into the incubator supporting use cases
- Milestone 3: Unit tests exist, work, and are checked into SVN.
- Milestone 4: Write documentation.

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

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

7. Class Index

- Zend_Filter_Input

8. Use Cases

UC-A

General usage.

```php
<?php

error_reporting( E_ALL | E_STRICT );

require_once 'Zend/Filter/Input.php';
require_once 'Zend/Validate/Between.php';

// Declare filter rules
$filters = array(

    // Chain of two filters specified by name
    'month' => array('digits', 'StringTrim'),

);

// Declare validator rules
$validators = array(

    // Simple field => validator by name
    'product' => 'alpha',

    // Chain of two validators, one by name one by object
    'month' => array(
        'digits', new Zend_Validate_Between(1, 12)
    ),

    // No validator, just presence is required
    'version' => array(
        'presence' => 'required'
    )

);

// Set up sample data (typically this would be $_GET or $_POST)
$data = array(
    'product' => 'ZendFramework', // field with one validator
    'month'   => '6',             // field with two filters and two validators
    'foo'     => 'bar'            // 'foo' field is unknown, not declared in validators
                                  // 'version' field is required but missing
);

// Process the inputs against the filters and validators.
$input = new Zend_Filter_Input($filters, $validators, $data);

// None in sample data above
if ($input->hasInvalid()) {
    echo "Invalid fields:\n";
    print_r($input->getInvalid());
    echo "\n";
}

// Should report field 'foo' is unknown
if ($input->hasUnknown()) {
    echo "Unknown fields:\n";
    print_r($input->getUnknown());
    echo "\n";
}

// Should report field 'version' is missing
if ($input->hasMissing()) {
    echo "Missing fields:\n";
    print_r($input->getMissing());
    echo "\n";
}

// Get validated fields, escaped by default escape filter (HtmlEntities)
echo "Product = " . $input->product . "\n";
echo "Month = " . $input->month . "\n";
```

UC-B

Specify default escape filter.

```
...

// Use Zend_Filter_Trim instead of default Zend_Filter_HtmlEntities
$input = new Zend_Filter_Input($filters, $validators, $data, array('escape_filter' => 'trim'));

// Get product field with trim filter applied.
echo "Product = " . $input->product . "\n";

// Another way to get the same field.
echo "Product = " . $input->getEscaped('product') . "\n";
```

UC-C

Specify custom namespace to find user-defined filter and validator classes.

```
...

// Find 'My_Filter_Foo' if 'Zend_Filter_Foo' is not found.
// User-defined classes must still implement Zend_Filter_Interface or Zend_Validate_Interface.

$input = new Zend_Filter_Input($filters, $validators, $data, array('namespace' => 'My_Filters'));

// Add another namespace (still haven't processed input against filters and validators yet):
$input->addNamespace('Third_Party_Filters');

...
```

UC-01

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-01 given in that proposal.

Basic example:

```
$filters = array(
    'phone_no'    => 'digits',
    'description' => 'stripTags'
);
$validators = array(
    'phone_no'    => 'digits',
    'description' => new Zend_Validate_StringLength(255)
);
$input = new Zend_Filter_Input($filters, $validators, $post);
if (!$input->hasInvalid()) {
    ...
}
```

UC-02

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-02 given in that proposal.

Globbing:

```
$filters = array(
    '*' => array('htmlEntitiesDecode', 'stripTags')
);
$validators = array(
    '*' => new Zend_Validate_Regex('/./')
);
$input = new Zend_Filter_Input($filters, $validators, $post);
if (!$input->hasInvalid()) {
    ...
}
```

Grouping; Zend_Filter_Input doesn't do grouping as Zend_Validate_Builder does, so this is how one would have to achieve the same result:

```
$filters = array(
    'phone_no' => 'digits',
    'ss_no'    => 'digits',
    'cc_no'    => 'digits'
);
$strlen = new Zend_Validate_StringLength(100);
$validators = array(
    'address1' => $strlen,
    'address2' => $strlen,
    'address3' => $strlen,
);
$input = new Zend_Filter_Input($filters, $validators, $post);
if (!$input->hasInvalid()) {
    ...
}
```

UC-03

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-03 given in that proposal.

Password confirmation:

```
$validators = array(
  'password1' => 'Password',
  'password2' => 'Password',
  'passwords' => array(
    'stringEquals', // proposing new validate class
    'fields' => array('password1', 'password2')
  )
);
$input = new Zend_Filter_Input(null, $validators, $post);
$input->addNamespace('My_Validate'); // to find My_Validate_Password class
if (!$input->hasInvalid()) {
    ...
}
```

Captcha validation:

```
$validators = array(
  'captcha' => new Zend_Validate_Equals($session['captcha']) // proposing new validate class
);
$input = new Zend_Filter_Input(null, $validators, $post);
if (!$input->hasInvalid()) {
    ...
}
```

UC-04

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in

UC-04 given in that proposal.

UC-05

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-05 given in that proposal.

UC-06

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-06 given in that proposal.

UC-07

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-07 given in that proposal.

UC-08

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-08 given in that proposal.

UC-09

For comparison to the Zend_Validate_Builder proposal, below is example code using Zend_Filter_Input that implement the solution described in UC-09 given in that proposal.

9. Class Skeletons

Zend_Filter_Input

```
<?php

require_once 'Zend/Loader.php';
require_once 'Zend/Filter.php';
require_once 'Zend/Validate.php';

class Zend_Filter_Input
{

    const OPT_ESCAPE_FILTER = 'escape_filter';
    const OPT_NAMESPACE     = 'namespace';

    const RULE              = 'rule';
    const FIELD             = 'field';
    const PRESENCE          = 'presence';
    const VALIDATOR         = 'validator';
    const VALIDATOR_CHAIN   = 'validatorChain';
    const BREAK_CHAIN       = 'breakChainOnFailure';

    const PRESENCE_REQUIRED = 'required';
    const PRESENCE_OPTIONAL = 'optional';
```

```
    /**
     * @var array
     */
    protected $_originalData = array();

    /**
     * @var array
     */
    protected $_data = array();

    /**
     * @var array
     */
    protected $_filterRules = array();

    /**
     * @var array
     */
    protected $_validatorRules = array();

    /**
     * @var array
     */
    protected $_validFields = array();

    /**
     * @var array
     */
    protected $_invalidFields = array();

    /**
     * @var array
     */
    protected $_missingFields = array();

    /**
     * @var array
     */
    protected $_unknownFields = array();

    /**
     * @var array
     */
    protected $_namespaces = array('Zend_Filter', 'Zend_Validate');

    /**
     * @var array
     */
    protected $_userNamespaces = array();

    /**
     * @var Zend_Filter_Interface
     */
    protected $_defaultEscapeFilter = null;

    /**
     * @var boolean
     */
    protected $_processed = false;

    /**
     * @param array $filters
     * @param array $validators
     * @param array $data       OPTIONAL
     * @param array $options    OPTIONAL
     */
    public function __construct(array $filterRules, array $validatorRules, array $data = null, array
$options = null)
    {
```

```
        if ($options) {
            $this->setOptions($options);
        }

        $this->_filterRules = $filterRules;
        $this->_validatorRules = $validatorRules;

        if ($data) {
            $this->setData($data);
        }
    }

    /**
     * @param mixed $namespaces
     * @return void
     */
    public function addNamespace($namespaces)
    {
        foreach((array) $namespaces as $namespace) {
            $this->_userNamespaces[Zend_Db_Adapter_Odbtp_Mssql] = $namespace;
        }
        $this->_namespaces = array_merge($this->_userNamespaces, array('Zend_Filter',
'Zend_Validate'));
    }

    /**
     * @return boolean
     */
    public function hasInvalid()
    {
        $this->_process();
        return !(empty($this->_invalidFields));
    }

    /**
     * @return boolean
     */
    public function hasMissing()
    {
        $this->_process();
        return !(empty($this->_missingFields));
    }

    /**
     * @return boolean
     */
    public function hasUnknown()
    {
        $this->_process();
        return !(empty($this->_unknownFields));
    }

    /**
     * @return array
     */
    public function getInvalid()
    {
        $this->_process();
        return $this->_invalidFields;
    }

    /**
     * @return array
     */
    public function getMissing()
    {
        $this->_process();
        return $this->_missingFields;
    }
```

```
    /**
     * @return array
     */
    public function getUnknown()
    {
        $this->_process();
        return $this->_unknownFields;
    }

    /**
     * @return string
     */
    public function getEscaped($fieldName)
    {
        $this->_process();
        $escapeFilter = $this->_getDefaultEscapeFilter();

        if (isset($this->_validFields[$fieldName])) {
            return $escapeFilter->filter($this->_validFields[$fieldName]);
        } else {
            return null;
        }
    }

    protected function _getDefaultEscapeFilter()
    {
        if ($this->_defaultEscapeFilter !== null) {
            return $this->_defaultEscapeFilter;
        }
        return $this->setDefaultEscapeFilter('htmlEntities');
    }

    /**
     * @return string
     */
    public function getUnescaped($fieldName)
    {
        $this->_process();
        if (isset($this->_validFields[$fieldName])) {
            return $this->_validFields[$fieldName];
        } else {
            return null;
        }
    }

    /**
     * @return string
     */
    public function __get($fieldName)
    {
        return $this->getEscaped($fieldName);
    }

    /**
     * @param string $fieldName
     * @return boolean
     */
    public function __isset($fieldName)
    {
        $this->_process();
        return isset($this->_validFields[$fieldName]);
    }

    /**
     * @param array $data
     * @return void
     */
    public function setData(array $data)
    {
        $this->_originalData = $data;
```

```
        $this->_data = $data;

        // Reset to initial state
        $this->_validFields = array();
        $this->_invalidFields = array();
        $this->_missingFields = array();
        $this->_unknownFields = array();

        $this->_processed = false;
    }

    /**
     * @param mixed $escapeFilter
     * @return void
     */
    public function setDefaultEscapeFilter($escapeFilter)
    {
        if (is_string($escapeFilter)) {
            $escapeFilter = $this->_getFilter($escapeFilter);
        }
        if (!$escapeFilter) {
            require_once 'Zend/Filter/Exception.php';
            throw new Zend_Filter_Exception('Cannot find escape filter');
        }
        $this->_defaultEscapeFilter = $escapeFilter;
        return $escapeFilter;
    }

    /**
     * @param array $options
     * @return void
     * @throws Zend_Filter_Exception if an unknown option is given
     */
    public function setOptions(array $options)
    {
        foreach ($options as $option => $value) {
            switch (strtolower($option)) {
                case OPT_ESCAPE_FILTER:
                    $this->setDefaultEscapeFilter($value);
                    break;
                case OPT_NAMESPACE:
                    $this->addNamespace($value);
                    break;
                default:
                    require_once 'Zend/Filter/Exception.php';
                    throw new Zend_Filter_Exception("Unknown option '$option'");
                    break;
            }
        }
    }

    /*
     * Protected methods
     */

    /**
     * @return void
     */
    protected function _process()
    {
        if ($this->_processed === false) {
            $this->_processFilterRules();
            $this->_processValidatorRules();
            $this->_processed = true;
        }
    }

    /**
     * @return void
     * @throws Zend_Filter_Exception
```

```
     */
    protected function _processFilterRules()
    {
        foreach ($this->_filterRules as $ruleName => $filterRule) {
            if (!is_array($filterRule)) {
                $filterRule = array($filterRule);
            }

            $filterList = array();
            foreach ($filterRule as $key => $value) {
                if (is_int($key)) {
                    $filterList[Zend_Db_Adapter_Odbtp_Mssql] = $value;
                }
            }

            if (!isset($filterRule[self::FIELD])) {
                $filterRule[self::FIELD] = $ruleName;
            }

            $filterChain = new Zend_Filter();
            foreach ($filterList as $filter) {
                if (is_string($filter)) {
                    $filter = $this->_getFilter($filter);
                }
                if (!($filter && $filter instanceof Zend_Filter_Interface)) {
                    require_once 'Zend/Filter/Exception.php';
                    throw new Zend_Filter_Exception('Expected object implementing
Zend_Filter_Interface, got '.get_class($filter));
                }
                $filterChain->addFilter($filter);
            }

            $field = $filterRule[self::FIELD];

            // @todo: support multi-valued data inputs
            $this->_data[$field] = $filterChain->filter($this->_data[$field]);
        }
    }

    /**
     * @return void
     * @throws Zend_Validate_Exception
     */
    protected function _processValidatorRules()
    {
        foreach ($this->_validatorRules as $ruleName => $validatorRule) {
            if (!is_array($validatorRule)) {
                $validatorRule = array($validatorRule);
            }

            $validatorList = array();
            foreach ($validatorRule as $key => $value) {
                if (is_int($key)) {
                    $validatorList[Zend_Db_Adapter_Odbtp_Mssql] = $value;
                }
            }

            // set defaults
            if (!isset($validatorRule[self::BREAK_CHAIN])) {
                $validatorRule[self::BREAK_CHAIN] = false;
            }
            if (!isset($validatorRule[self::FIELD])) {
                $validatorRule[self::FIELD] = $ruleName;
            }
            if (!isset($validatorRule[self::PRESENCE])) {
                $validatorRule[self::PRESENCE] = self::PRESENCE_OPTIONAL;
            }

            $validatorChain = new Zend_Validate();
            foreach ($validatorList as $validator) {
```

```
                if (is_string($validator)) {
                    $validator = $this->_getValidator($validator);
                }
                if (!($validator && $validator instanceof Zend_Validate_Interface)) {
                    require_once 'Zend/Validate/Exception.php';
                    throw new Zend_Validate_Exception('Expected object implementing
Zend_Validate_Interface, got '.get_class($validator));
                }
                $validatorChain->addValidator($validator, $validatorRule[self::BREAK_CHAIN]);
            }

            $field = $validatorRule[self::FIELD];

            if (!isset($this->_data[$field]) && $validatorRule[self::PRESENCE] ==
self::PRESENCE_REQUIRED) {
                $this->_missingFields[$field][Zend_Db_Adapter_Odbtp_Mssql] = "Field '$field' is
required by rule $ruleName, but field is missing.";
                continue;
            }

            // @todo: support multi-valued data inputs
            if (!$validatorChain->isValid($this->_data[$field])) {
                $this->_invalidFields[$field] = array_merge($this->_invalidFields,
$validatorChain->getMessages());
                continue;
            }

            $this->_validFields[$field] = $this->_data[$field];
        }

        /**
         * Unset fields in $_data that have been added to other arrays.
         * We have to wait until all rules have been processed because
         * a given field may be referenced by multiple rules.
         */
        foreach (array_merge(
            array_keys($this->_validFields),
            array_keys($this->_invalidFields),
            array_keys($this->_missingFields)) as $key) {
            unset($this->_data[$key]);
        }

        /**
         * Anything left over in $_data is an unknown field.
         */
        $this->_unknownFields = $this->_data;
    }

    /**
     * @param string $classBaseName
     * @return Zend_Filter_Interface or null
     */
    protected function _getFilter($classBaseName)
    {
        return $this->_getFilterOrValidator('Zend_Filter_Interface', $classBaseName);
    }

    /**
     * @param string $classBaseName
     * @return Zend_Validate_Interface or null
     */
    protected function _getValidator($classBaseName)
    {
        return $this->_getFilterOrValidator('Zend_Validate_Interface', $classBaseName);
    }

    /**
     * @param string $derivedFrom
     * @param string $classBaseName
     * @return mixed
```

```
 */
protected function _getFilterOrValidator($derivedFrom, $classBaseName)
{
    foreach ($this->_namespaces as $namespace) {
        $className = $namespace . '_' . ucfirst($classBaseName);
        try {
            Zend_Loader::loadClass($className);
            $object = new $className();
            if ($object instanceof $derivedFrom) {
                return $object;
            }
        } catch (Zend_Exception $e) {
            // fallthrough and continue
        }
    }
    return null;
```

```
        }

    }
```

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