Zend_Filter static usage - 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 static usage Component Proposal

Proposed Component Name	Zend_Filter static usage
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Filter static usage
Proposers	Bill Karwin Darby Felton
Revision	1.1 - 12 April 2007: initial writeup. (wiki revision: 7)

Table of Contents

1. Overview
2. References
3. Requirements
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Skeletons
10. Testing Strategy

1. Overview

This is a proposed solution for a convenient static method to allow Zend_Filter and Zend_Validate to work in an ad-hoc fashion.

This proposal is intended to **supplement**, not to replace the current object-oriented usage of Zend_Filter and Zend_Validate. You would still create instances of classes that implement Zend_Filter_Interface and Zend_Validate_Interface and use them as objects, call their `isValid()` and `filter()` methods, etc.

2. References

- tbd

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** create an static interface to filtering and validating.
- This component **will** load, instantiate, and invoke the Filter or Validate objects.
- 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_Validate_Interface
- Zend_Validate_Exception
- other concrete classes that implement Zend_Filter_Interface and Zend_Validate_Interface

5. Theory of Operation

The `call()` method in classes Zend_Filter and Zend_Validate provide a static interface to invoke a single class that implement Zend_Filter_Interface or Zend_Validate_Interface respectively in their default usage. The desired filter or validator is specified by the class basename.

```
$newValue = Zend_Filter::call($oldValue, 'filtername');
```

In the example above, 'filtername' is transformed to 'Zend_Filter_Filtername' and an instance is created. Then its `filter()` method is called on the `$oldValue` provided and the value returned.

If no such filter class can be found, an exception is thrown.

Some filters take arguments in their constructor. Pass these arguments as an array in the third argument to the `call()` method.

```
$newValue = Zend_Filter::call($oldValue, 'filtername', array('foo', 'bar'));
```

You can make the `call()` method search for classes in other namespaces than Zend_Filter and Zend_Validate, by passing a string or array as the fourth argument. User-specified namespaces are searched before Zend_Filter or Zend_Validate namespaces.

```
$newValue = Zend_Filter::call($oldValue, 'filtername', array(), array('My_Filters',
'Third_Party_Filters'));
```

If the filter or validator takes no arguments, you must pass an empty array. A `null` or `false` is interpreted as an argument with that value to pass to the constructors.

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
- Zend_Validate

8. Use Cases

UC-01

Apply a single Filter to a single value.

```
<?php
error_reporting( E_ALL | E_STRICT );
require_once 'Zend/Filter';

$string =

$string = Zend_Filter::call($string, 'htmlEntities');

?>
```

UC-02

Apply a single Validator to a single value.

```
<?php
error_reporting( E_ALL | E_STRICT );
require_once 'Zend/Validate.php';

echo '6 is between 1 and 12? ';

$isBetween = Zend_Validate::call(6, 'between', array(1, 12));

echo ($isBetween ? 'yes' : 'no') . "\n";

?>
```

UC-03

Use a user-defined validate class in a different namespace.

```
<?php
error_reporting( E_ALL | E_STRICT );
require_once 'Zend/Validate.php';

// Must pass empty array for validator options.
$bool = Zend_Validate::call($value, 'myValidator', array(), 'My_Validators');
```

9. Class Skeletons

Zend_Filter

The following method is added to the Zend_Filter class. The existing methods of that class are not changed.

```
/**
 * @param mixed    $value
 * @param string   $classBaseName
 * @param array    $args          OPTIONAL
 * @param mixed    $userPackages  OPTIONAL
 * @return boolean
 * @throws Zend_Filter_Exception
 */
public static function call($value, $classBaseName, array $args = array(), $userPackages = array())
{
    $packages = array_merge((array) $userPackages, array('Zend_Filter'));
    foreach ($packages as $package) {
        $className = $package . '_' . ucfirst($classBaseName);
        try {
            require_once 'Zend/Loader.php';
            Zend_Loader::loadClass($className);
            $class = new ReflectionClass($className);
            $filter = $class->newInstanceArgs((array) $args);
            if ($filter instanceof Zend_Filter_Interface) {
                return $filter->filter($value);
            }
        } catch (Zend_Exception $ze) {
            // fallthrough and continue
        }
    }
    throw new Zend_Filter_Exception("Filter class not found from basename '$classBaseName'");
}
```

Zend_Validate

The following method is added to the Zend_Validate class. The existing methods of that class are not changed.

```
/**
 * @param mixed    $value
 * @param string   $classBaseName
 * @param array    $args          OPTIONAL
 * @param mixed    $userPackages  OPTIONAL
 * @return boolean
 * @throws Zend_Validate_Exception
 */
public static function call($value, $classBaseName, array $args = array(), $userPackages = array())
{
    $packages = array_merge((array) $userPackages, array('Zend_Validate'));
    foreach ($packages as $package) {
        $className = $package . '_' . ucfirst($classBaseName);
        try {
            require_once 'Zend/Loader.php';
            Zend_Loader::loadClass($className);
            $class = new ReflectionClass($className);
            $validator = $class->newInstanceArgs((array) $args);
            if ($validator instanceof Zend_Validate_Interface) {
                return $validator->isValid($value);
            }
        } catch (Zend_Exception $ze) {
            // fallthrough and continue
        }
    }
    throw new Zend_Validate_Exception("Validate class not found from basename '$classBaseName'");
}
```

10. Testing Strategy

To be written.

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