Zend_Markup - Pieter Kokx

<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_Markup Component Proposal

Proposed Component Name	Zend_Markup
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Markup
Proposers	Pieter Kokx
Zend Liaison	Ralph Schindler
Revision	1.1 - 31 January 2008: Created the proposal. 1.2 - 4 February 2008: Finished the proposal and submitted for community review. 1.3 - 25 April 2008: Changed a use case. 1.4 - 26 April 2008: Added the source code. 1.5 - 5 June 2008: Added link to subversion repository. 2.0 - 9 August 2008: Refactored Zend_TextParser to Zend_Markup 2.1 - 25 October 2008: Submitted the proposal for community review (wiki revision: 38)

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

Zend_Markup should provide an extensible way to tokenize and render lightweight markup languages, like BBcode and Textile.

2. References

- BBcode Wikipedia
- Lightweight markup language Wikipedia

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** provide the extensibility to tokenize different lightweight markup languages.

- This component **will** provide the extensibility to render into different (lightweight) markup languages.
- This component **will** provide an easy way to create your own tags.
- It **will not** be possible to retrieve the original source from the rendered text.

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Filter
- Zend_Loader_PluginLoader

5. Theory of Operation

Zend_Markup does tokenize and render lightweight markup languages into another format. Because there are a lot of lightweight markup languages, it should be compatible with the most important languages. It should also be possible to create your own Parsers.

Zend_Markup consists of parsers and renderers. A parser is splitting the input text into an array with all the information it could extract out of the input text.

For example, the Zend_Markup_Parser_Bbcode parser should produce this array from the string '[tag="a" attr=val]value[/tag]':

```
array(
    array(
        'tag'        => '[tag="a" attr=val]',
        'type'       => Zend_Markup::TYPE_TAG,
        'name'       => 'tag',
        'stoppers'   => array('[/]', '[/tag]'),
        'attributes' => array(
            'tag'  => 'a',
            'attr' => 'val'
        )
    ),
    array(
        'tag'   => 'value',
        'type'  => Zend_Markup::TYPE_NONE
    ),
    array(
        'tag'        => '[/tag]',
        'type'       => Zend_Markup::TYPE_STOPPER,
        'name'       => 'tag',
        'stoppers'   => array(),
        'attributes' => array()
    )
);
```

A renderer does loop trough the generated array, and uses the provided information to generate a value.

6. Milestones / Tasks

- Milestone 1: [DONE] Create the proposal
- Milestone 2: [DONE] Initial class design
- Milestone 3: [DONE] Submit the proposal for community review
- Milestone 4: [DONE] Create working prototype (see the bottom of this proposal)
- Milestone 5: [DONE] Create code-covering unit tests.

7. Class Index

- Zend_Markup
- Zend_Markup_Parser_Interface
- Zend_Markup_Parser_BbCode
- Zend_Markup_Parser_Textile
- Zend_Markup_Parser_Parsed
- Zend_Markup_Renderer_Abstract
- Zend_Markup_Renderer_Html
- Zend_View_Helper_Markup

8. Use Cases

UC-01

Simple usage (bbcode)

```
<?php
$bbcode = Zend_Markup::factory('BbCode', 'Html');
// instance of Zend_Markup_Renderer_Html, with Zend_Markup_Parser_BbCode as its parser

// should output XHTML valid output: '<strong><em>some text</em></strong><em>more text</em> some text'
echo $bbcode->render('[b][i]some text[/b]more text[/i] some text');
```

UC-02

Creating your own tags (bbcode)

```
<?php
// instantiate a bbcode object like above

// the tag is allowed to be inside this tags
$allowedIn    = array('i', 'u', 's');
// the tag allows this tags to occur inside it
$allowsInside = array('i', 'u', 's');

$bbcode->addTag('b', Zend_Markup::REPLACE, array('start' => '<strong>', 'end' => '</strong>'),
$allowedIn, $allowsInside);

// creating your own tags should work with any format that supports custom tags

// should output: <strong>some text</strong>
echo $bbcode->render('[b]some text[/b]');
```

UC-03

Create an object with a textile parser (uses the same renderer as with the BBcode parser).

```
<?php
$wiki = Zend_Markup::factory('Textile', 'Html');

// should output: <strong>some text</strong>
echo $wiki->render("*some text*");
```

UC-04

Filtering (bbcode)

```
<?php
// instantiate a bbcode object like above

require_once 'Zend/Filter/Word/SeparatorToCamelCase.php';

$renderer->addFilter(new Zend_Filter_Word_SeparatorToCamelCase());

// should output: SomeText<code>some text</code> (filtering is disabled in the [code] tag)
echo $bbcode->render('some text[code]some text[/code]');
```

9. Class Skeletons

Zend_Markup_Parser_Interface

```php
<?php
interface Zend_Markup_Parser_Interface
{

    /**
     * Parse a string
     *
     * This should output something like this:
     *
     * <code>
     * array(
     *     array(
     *         'tag'        => '[tag="a" attr=val]',
     *         'type'       => Zend_Markup::TYPE_TAG,
     *         'name'       => 'tag',
     *         'stoppers'   => array('[/]', '[/tag]'),
     *         'attributes' => array(
     *             'tag'  => 'a',
     *             'attr' => 'val'
     *         )
     *     ),
     *     array(
     *         'tag'   => 'value',
     *         'type'  => Zend_Markup::TYPE_NONE
     *     ),
     *     array(
     *         'tag'        => '[/tag]',
     *         'type'       => Zend_Markup::TYPE_STOPPER,
     *         'name'       => 'tag',
     *         'stoppers'   => array(),
     *         'attributes' => array()
     *     )
     * )
     * </code>
     *
     * @param string $value
     *
     * @return array
     */
    public function parse($value);
}
```

Zend_Markup_Renderer_Abstract

```
abstract class Zend_Markup_Renderer_Abstract
{

    /**
     * The parser
     *
     * @var Zend_Markup_Parser_Interface
     */
    protected $_parser;


    /**
     * Set a parser
     *
     * @param Zend_Markup_Parser_Interface $parser
     *
     * @return Zend_Markup_Renderer_Abstract
     */
    public function setParser(Zend_Markup_Parser_Interface $parser)
    {
        $this->_parser = $parser;

        return $this;
    }

    /**
     * Get a parser
     *
     * @return Zend_Markup_Parser_Interface
     */
    public function getParser()
    {
        return $this->_parser;
    }

    /**
     * Render a value
     *
     * @param string $value
     *
     * @return string
     */
    abstract public function render($value);
}
```

Zend_Markup

```


class Zend_Markup
{
    const REPLACE = 'replace';
    const REPLACE_SINGLE = 'replace_single';
    const CALLBACK = 'callback';
    const CALLBACK_SINGLE = 'callback_single';

    const TYPE_NONE    = 'none';
    const TYPE_TAG     = 'tag';
    const TYPE_STOPPER = 'stopper';
    const TYPE_SINGLE  = 'single';

    /**
     * The parser loader
     *
     * @var Zend_Loader_PluginLoader
     */
    protected static $_parserLoader;
```

```
    /**
     * The renderer loader
     *
     * @var Zend_Loader_PluginLoader
     */
    protected static $_rendererLoader;


    /**
     * Get the parser loader
     *
     * @return Zend_Loader_PluginLoader
     */
    public static function getParserLoader()
    {}

    /**
     * Get the renderer loader
     *
     * @return Zend_Loader_PluginLoader
     */
    public static function getRendererLoader()
    {}

    /**
     * Add a parser path
     *
     * @param string $prefix
     * @param string $path
     *
     * @return Zend_Loader_PluginLoader
     */
    public static function addParserPath($prefix, $path)
    {}

    /**
     * Add a renderer path
     *
     * @param string $prefix
     * @param string $path
     *
     * @return Zend_Loader_PluginLoader
     */
    public static function addRendererPath($prefix, $path)
    {}

    /**
     * Factory pattern
     *
     * @param string $parser
     * @param string $renderer
     * @param array $options
     *
     * @return Zend_TextParser_Renderer_Abstract
     */
```

```
    public static function factory($parser, $renderer, array $options = array())
    {}
}
```

Zend_View_Helper_Markup

```
class Zend_View_Helper_Markup
{

    /**
     * The parser to use
     *
     * @var string
     */
    protected $_parser = 'bbCode';

    /**
     * The renderer to use
     *
     * @var string
     */
    protected $_renderer = 'html';

    /**
     * Zend_Markup instance
     *
     * @return void
     */
    protected $_markup;


    /**
     * View wrapper for Zend_Markup
     *
     * @return Zend_View_Helper_Markup|string
     */
    public function markup($value = null, $parser = 'bbCode', $renderer = 'html')
    {}

    /**
     * Set the Zend_Markup_Renderer_Abstract instance
     *
     * @param Zend_Markup_Renderer_Abstract $instance
     *
     * @return Zend_View_Helper_Markup
     */
    public function setTextParser(Zend_Markup_Renderer_Abstract $instance)
    {}

    /**
     * Get the Zend_Markup_Renderer_Abstract instance
     *
     * @return Zend_Markup_Renderer_Abstract
     */
    public function getMarkup()
    {}
}
```

You can retrieve the current code from the incubator.
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>