Zend_View Enhanced - Pádraic Brady

<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_View Enhanced Component Proposal

Proposed Component Name	Zend_View Enhanced
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_View Enhanced
Proposers	Pádraic Brady Matthew Weier O'Phinney, Zend liaison
Revision	1.2 - 26 June 2007 (wiki revision: 25)

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

The purpose of the Zend_View Enhanced proposal is to propose the addition of Zend_View_Abstract methods, and Zend_View_Helper_* classes, which in combination provide for the ability to generate output in a flexible manner which adheres to the separation of Controller and View, advocates decoupling, and maintains backwards compatibility with the 1.0.0GA version of the Framework, and subsequent minor releases.

At present, Zend_View is a strict implementation of the Template View design pattern. It excels at templating, but has no native functionality for handling several key requirements of the View. This typically includes decorating templates with a common layout, supporting the rendering of templates in their own independent variable scope, allowing templates to transport additional presentation logic into other templates transparently, making calls into the Controller for additional embeddable output, and capturing common presentation logic as View Helpers. Suggested terms for these features:

1. Layouts
2. Partials
3. Controllers
4. Placeholders
5. View Helpers (as now)

The common perception is that the Controller is required to facilitate many of these View features. Unfortunately, this is misleading. These approaches have no such dependency on the Controller unless such a dependency is implemented purposefully. All of the above are already standard for templating engines in several frameworks - even Smarty has native capabilities for many of these. The Controller is not a requirement for their implementation.

The argument this idea springs from is that Controller based solutions inevitably require far more code. This in turn requires more processing (and a possible performance hit), and the writing of a multitude of additional Controllers. Controller based approaches usually violate KISS (Keep It Simple Stupid) and are not a replacement for View based functions which are available as part and parcel of the template syntax.

The proposal, however, makes no assumptions about the Controller. The only such assumption is that a minor issue may be resolved (#ZF-1545). If no resolution is forthcoming, an alternate means of implementation may be used.

Wherever possible, View Helpers are preferred in place of adding code directly to Zend_View_Abstract. A number of these View Helpers have been proposed which capture the above four concepts, and add specialised variants to simplify template authoring. These expand the reach of View Helpers primarily to the <head> section of a HTML document where a lot of activity around Javascript, CSS, meta tags, etc. may be controlled outside a Layout within the actual sub-templates (a common requirement of any nested View design).

The proposed additions maintain Zend_View's decoupling from Zend_Controller. They should (as a requirement) have no conflict with Controller integrated strategies for similar features. ~~There are therefore no conflicts with the addition of Controller based variants for Layouts such as the introduction of a Two-Step View implementation.~~ However, layouts will be dealt with under a separate proposal.

The additions are therefore quite straight forward, easy to implement, and pose no threat to backwards compatibility.

2. References

- Sample implementation in Subversion (this source code is not a complete implementation as this proposal has yet to be reviewed)
- Relevant blog post discussing the concepts
- Relevant blog post discussing Zend_View Enhanced
- Symfony Documentation for its View Layer
- The venerable Smarty library (much of the above is already possible using plain old Smarty)
- The Struts2 Tiles Tag Library

3. Component Requirements, Constraints, and Acceptance Criteria

- **Must** implement the -4-3 view construction strategies (~~Layout,~~ Partial, Controller, Placeholder) *(Note: Layouts will be dealt with in a separate proposal)*
- **Must** implement several specialised View Helpers for controlling Element additions to the <head>
- **Must** maintain backwards compatibility with Zend_View and Zend_Controller_Action_Helper_ViewRenderer
- **Must** (where feasible) be implemented as default View Helpers
- **Should not** prevent or restrict Controller derived strategies for output construction
- **Must** be accompanied by a full set of integration tests prior to acceptance in Core
- **May optionally require** adoption of the setters suggested in Zend_Font - Karol Babioch#ZF-1545

4. Dependencies on Other Framework Components

- Zend_View_Abstract
- Zend_View_Exception
- Zend_Controller_* (only for Controller strategy)

5. Theory of Operation

Zend_View Enhanced advocates a template driven approach to output generation. As templates are rendered by Zend_View, the templates themselves may invoke Partials and Controllers, as well as set Placeholder content (template specific or default) which is centrally stored for higher level and subsequent templates (such as Layouts) to utilise. ~~Layouts are applied at the end of any Zend_View instance's rendering process to take advantage of Placeholders, and their specialised variants.~~

Partials: Partials simply render a specified template within it's own separate variable scope. This is useful for reusable template fragments, avoids variable name clashes, and allows configuration of Partials to include from independent Modules (i.e. using that Module's helpers/filters/paths). At a higher level, Partials implement the Composite View design pattern allowing for the generation of a tree of View objects for even more flexibility.

Controllers: Dispatches a request to the Controller which returns output for embedding in a template. This must be compatible with ViewRenderer. May be used for querying the Model, but it's generally recommended to use a View Helper where possible. Most likely useful where authentication or authorisation must also be queried before Model access, or where existing Controllers can be reused as an alternative to implementing more View Helpers.

~~Layouts: Decorates the main output of a Zend_View instance with a Layout template composed of common page elements and Placeholders for context specific additions which are defined by templates prior to the Layout rendering. Layouts in Zend_View are implemented in a simple manner - they are the only feature requiring editing of Zend_View_Abstract.~~

Placeholders: Allows templates to set centrally registered content for inclusion in any subsequently rendered template or Layout. Placeholders are sufficiently decoupled from templates so that templates from different variable scopes can still communicate data to each other. Set Placeholders are only ever available to templates rendered after the Placeholder is set. Placeholders may be used for HTML elements which require a strict order of addition (e.g. Javascript <script> tags), or alternatively a specialised HTML generating Helper (e.g. Zend_View_Helper_HeadScript) can be used instead. See next.

HTML <head> View Helpers: Utilise Placeholders in a pre-defined namespace to dynamically control the construction of a Layout's <head> section. This eases the common use cases of using sub-templates to dynamically define additional Javascript, CSS, Metas, and other such elements for inclusion in a Layout.

6. Milestones / Tasks

- Milestone 1: Write unit tests capturing the agreed interface and behaviour
- Milestone 2: Write the code required to pass all unit tests
- Milestone 3: Write integration tests to cover various edge cases
- Milestone 4: Verify that code does not compromise backwards compatibility with ZF 1.0.0GA
- Milestone 5: Complete documentation

7. Class Index

- Zend_View_Abstract
- Zend_View_Factory
- Zend_View_Helper_Placeholder
- Zend_View_Helper_Controller
- Zend_View_Helper_Partial
- Zend_View_Helper_HeadTitle
- Zend_View_Helper_HeadScript
- Zend_View_Helper_HeadLink
- Zend_View_Helper_HeadMeta
- Zend_View_Helper_HeadStyle
- Zend_View_Helper_Doctype

8. Use Cases

UC-01

Partial Use Case

Partials render a template fragment at a specific location within another template. They are rendered in a separate variable scope with a Model supplied by the parent template. It is assumed Partials can therefore be easily rendered from different Modules.

Partials implement the Composite View Pattern defined in J2EE Design Patterns. Although template fragment rendering is likely the majority use case, partials are infinitely nestable and can thus compose a nested tree of Zend_View objects for above average complex Views.

Since the primary use case of partials is for use in loops, an additional helper, partialLoop, will be created. This will take the passed array and loop over it, passing each element to the partial view script requested.

./src/default/views/scripts/blog/index.phtml

```
<?php echo $this->render('stdHeader.phtml') ?>

    <div class="blog content">
        <?php foreach($this->entries as $entry): ?>
            <?php echo $this->partial('_entry.phtml', array('entry'=>$entry)) ?>
        <?php endforeach; ?>
    </div>

<?php echo $this->render('stdFooter.phtml') ?>
```

./src/default/views/scripts/blog/_entry.phtml

```
<div class="entry">
    <ul class="meta">
        <li><?php echo $this->entry->author; ?></li>
        <li><?php echo $this->entry->date; ?></li>
        <li><?php echo $this->entry->comments; ?></li>
    </ul>
    <div class="entry body">
        <?php echo $this->entry->text; ?>
    </div>
</div>
```

./src/default/views/scripts/blog/index_alt.phtml

```
<?php echo $this->render('stdHeader.phtml') ?>

    <div class="blog content">
        <?php /* same thing as above, but instead of putting loop in view
               * script, handled in the helper
               */ ?>
        <?php echo $this->partialLoop('_entry.phtml', $this->entries) ?>
    </div>

<?php echo $this->render('stdFooter.phtml') ?>
```

UC-02

Controller Use Case

The Controller View Helper enables templates to dispatch any given Controller and Action. The contents of the Response object resulting from this dispatch are then rendered in the calling template. Controller should never be used to alter the Model in this fashion - but are highly usable to enable the embedding of an existing Controller's output into an application.

Note: actions may either _forward() to another action or _redirect(). These are considered invalid responses, and result in an empty string being returned. Developers should be careful to only invoke actions that do not _forward() or _redirect().

./src/default/views/scripts/blog/index.phtml

```
<?php echo $this->render('stdHeader.phtml'); ?>

    <div id="content">
        Some content of main body in page...
    </div id="content">
    <div id="sidebar right">
        <?php echo $this->controller('list', 'comment', null, array('count'=>10)); ?>
    </div>

<?php echo $this->render('stdFooter.phtml'); ?>
```

UC-03

Placeholder Use Case

Placeholders are centrally registered pieces of content which all subsequently rendered templates may access. Since Layouts are always rendered last - they may access all set Placeholders. Placeholders may be set with a specific order attached. This is beyond the scope of the the following Use Case which does not differentiate between <head> descendant elements.

Placeholders can either hold scalars or collections. In the case of collections, placeholders provide a mechanism for ordering and sorting. In either case, a placeholder would allow specifying content with which to either prefix or postfix the placeholder content, and in the case of collections, separator content. They will implement __toString(), allowing them to be echo'd directly. Finally, they will also allow capturing template content into a placeholder.

./src/default/view/scripts/layout.phtml

```
<?php echo $this->doctype('XHTML 1.0 Strict') ?>
<html>
<head>
    <?php echo $this->headTitle() ?>
    <link rel="stylesheet" type="text/css" href="styles/base.css" media="screen">
    <?php echo $this->headLink() ?>
</head>
<body>
<div id="nav">
    <?php echo $this->placeholder('subnav') ?>
</div>
<div id="main">
    <?php echo $this->layout_content; ?>
</div>
<div id="footer">
    <p>
    <?php if (count($this->placeholder('license')): ?>
        <?php echo $this->placeholder('license'); ?>
    <?php else: ?>
        Licensed under the <a href="http://www.gnu.org/licenses/fdl.html">GNU Free Documentation
License</a>.
    <?php endif; ?>
    </p>
    <p><small>Copyright &copy; 2007 Pádraic Brady</small></p>
</div>
</body>
</html>
```

./src/default/view/scripts/blog/index.phtml

```
<?php $this->headLink(array('rel'=>'alternate', 'type'=>'application/rss+xml', 'title'=>'RSS',
'href'=>'http://www.planet-php.org/rss/')); ?>
<?php $this->headLink(array('rel'=>'stylesheet', 'type'=>'text/css', 'href'=>'/style/extras.css')); ?>
<?php $this->placeholder('license')->set('Licensed under a <a
href="http://creativecommons.org/licenses/by/1.0/">Creative Commons License</a>.'); ?>

<div class="blog content">
    <?php foreach($this->entries as $entry): ?>
        <?php echo $this->partial('_entry.phtml', array('entry'=>$entry)) ?>
    <?php endforeach; ?>
</div>

<?php $this->placeholder('subnav')->captureStart(); ?>
<ul id="subnav">
    <li><a href="/blog/tags">Tags</a></li>
    <li><a href="/blog/authors">Authors</a></li>
</ul>
<?php $this->placeholder('subnav')->captureEnd(); ?>
```

9. Class Skeletons

Zend_View_Helper_Partial

```
/**
 * Helper for rendering a template fragment in its own variable scope.
 *
 * @package    Zend_View
 * @subpackage Helpers
 * @copyright  Copyright (c) 2007 Pádraic Brady (http://blog.astrumfutura.com)
 * @license    New BSD
 */
class Zend_View_Helper_Partial {

    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null;

    /**
     * Renders a template fragment within a variable scope distinct from the
     * calling View object.
     *
     * @param string $name
     * @param string|array $module
     * @param array $model
     * @returns string $output
     */
    public function partial($name, $module = null, array $model = null)
    {}

    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_Partial
     */
    public function setView(Zend_View_Interface $view)
    {}

    /**
     * Clone the current View within resorting to a Factory call
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_Partial
     */
    protected function _cloneView(array $viewModel = null)
    {}

}
```

Zend_View_Helper_PartialLoop

```
/**
 * Helper for rendering a template fragment in its own variable scope, looping
 * over an array.
 *
 * @package    Zend_View
 * @subpackage Helpers
 * @copyright  Copyright (c) 2007 Pádraic Brady (http://blog.astrumfutura.com)
 * @license    New BSD
 */
class Zend_View_Helper_PartialLoop extends Zend_View_Helper_Partial {

    /**
     * Renders a template fragment within a variable scope distinct from the
     * calling View object.
     *
     * @param string $name
     * @param string|array $module
     * @param array $model
     * @returns string $output
     */
    public function partialLoop($name, $module = null, array $model = null)
    {
        if ((null === $model) && is_array($module)) {
            $model = $module;
        }
        if (null === $model) {
            return '';
        }

        $content = '';
        foreach ($model as $item) {
            $content .= $this->partial($name, $module, $model);
        }

        return $content;
    }
}
```

Zend_View_Helper_Controller

```
/**
 * Helper for embedding the response output of an existing Controller action.
 *
 * @package    Zend_View
 * @subpackage Helpers
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 */
class Zend_View_Helper_Controller
{
    public $defaultModule;
    public $dispatcher;
    public $front;
    public $request;
    public $response;

    /**
     * Constructor
     *
     * Initialize various helper objects and state.
     *
     * @return void
     */
    public function __construct()
    {
        $this->front         = Zend_Controller_Front::getInstance();
```

```
        $this->request       = clone $this->front->getRequest();
        $this->response      = clone $this->front->getResponse();
        $this->dispatcher    = clone $this->front->getDispatcher();
        $this->defaultModule = $this->front->getDefaultModule();
    }

    /**
     * Reset state
     *
     * @return void
     */
    public function resetObjects()
    {
        // Should probably create a 'clearUserParams()' method...
        $params = $this->request->getUserParams();
        foreach (array_keys($params) as $key) {
            $this->request->setUserParam($key, null);
        }

        $this->response->clearBody()
             ->clearHeaders()
             ->clearRawHeaders();
    }

    /**
     * Invoke a controller component
     *
     * @param  string $action Action name
     * @param  string $controller Controller name
     * @param  string $module Module name; defaults to default module name as registered in dispatcher
     * @param  array $params Additional parameters to pass in request object
     * @return string
     */
    public function controller($action, $controller, $module = null, array $params = array())
    {
        $this->resetObjects();
        if (null === $module) {
            $module = $this->defaultModule;
        }
        $this->request->setParams($params)
             ->setModuleName($module)
             ->setControllerName($controller)
             ->setActionName($action)
             ->setDispatched(true);

        $this->dispatcher->dispatch($this->request, $this->response);

        if (!$this->request->isDispatched()
            || $this->response->isRedirect())
        {
            // forwards and redirects render nothing
            return '';
        }
```

```
        return $this->response->getBody();
    }
}
```

Zend_View_Helper_Placeholder

By allowing templates within the Composite View tree (i.e. nested render() calls, partials etc.) access a central object which all View objects may access as a central Registry, it becomes possible for nested templates, irrespective of nest depth, to influence their top-level decorating Layout, or subsequently rendered templates, by setting values these may later refer to.

All values are stored in array objects. This enables Placeholders to maintain an indexed set of values, where the order of inclusion at a rendering point is determined by this index. Because it implements ArrayObject, any PHP array sorting function may be used to sort the placeholder values.

This strategy is built into a set of a pre-defined Placeholders for the <head> element of HTML documents in the proposed Zend_View_Helper_Head* helpers. More similar helpers may be proposed if their utility proves valuable. Their purpose is primarily to offer specialised usage of the basic Placeholder class using Proxies which are capable of outputting valid XHTML/HTML using the values set by the template authors, and optionally specific to a defined Doctype.

```
/**
 * Container for placeholder values
 */
class Zend_View_Helper_Placeholder_Container extends ArrayObject
{
    const SET    = 'set';
    const APPEND = 'append';

    protected $_prefix    = '';
    protected $_postfix   = '';
    protected $_separator = '';

    protected $_captureLock = false;
    protected $_captureType;

    /**
     * Set a single value
     *
     * @param  mixed $value
     * @return void
     */
    public function set($value)
    {
        $this->exchangeArray(array($value));
    }

    /**
     * Retrieve container value
     *
     * If single element registered, returns that element; otherwise,
     * serializes to array.
     *
     * @return mixed
     */
    public function getValue()
    {
        if (1 == count($this)) {
            return $this[0];
        }

        return $this->getArrayCopy();
    }

    /**
     * Set prefix for __toString() serialization
     *
     * @param  string $prefix
     * @return Zend_View_Helper_Placeholder_Container
```

```
     */
    public function setPrefix($prefix)
    {}

    /**
     * Retrieve prefix
     *
     * @return string
     */
    public function getPrefix()
    {}

    /**
     * Set postfix for __toString() serialization
     *
     * @param  string $postfix
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function setPostfix($postfix)
    {}

    /**
     * Retrieve postfix
     *
     * @return string
     */
    public function getPostfix()
    {}

    /**
     * Set separator for __toString() serialization
     *
     * Used to implode elements in container
     *
     * @param  string $separator
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function setSeparator($separator)
    {}

    /**
     * Retrieve separator
     *
     * @return string
     */
    public function getSeparator()
    {}

    /**
     * Start capturing content to push into placeholder
     *
     * @param  int $type How to capture content into placeholder; append or set
     * @return void
     * @throws Zend_View_Helper_Placeholder_Exception if nested captures detected
     */
    public function captureStart($type = Zend_View_Helper_Placeholder_Container::APPEND)
    {
        if ($this->_captureLock) {
            throw new Zend_View_Helper_Placeholder_Exception('Cannot nest placeholder captures for the
same placeholder');
        }

        $this->_captureLock = true;
        $this->_captureType = $type;
        ob_start();
    }

    /**
     * End content capture
     *
```

```
     * @return void
     */
    public function captureEnd()
    {
        $data = ob_get_flush();
        $this->_captureLock = false;
        switch ($this->_captureType) {
            case self::SET:
                $this->exchangeArray(array($data));
                break;
            case self::APPEND:
            default:
                $this[Zend_Font - Karol Babioch] = $data;
                break;
        }
    }

    /**
     * Serialize object to string
     *
     * @return string
     */
    public function __toString()
    {
        $items  = $this->getArrayCopy();
        $return = $this->getPrefix()
                . implode($this->getSeparator(), $items)
                . $this->getPostfix();
        return $return;
    }
}

/**
 * Helper for passing data between otherwise segregated Views. It's called
 * Placeholder to make its typical usage obvious, but can be used just as easily
 * for non-Placeholder things. That said, the support for this is only
 * guaranteed to effect subsequently rendered templates, and of course Layouts.
 * This class is utilised by a number of specialised helpers such as
 * Zend_View_Helper_HeadTitle
 */
class Zend_View_Helper_Placeholder
{
    /**
     * @var Zend_View_Interface
     */
    public $view;

    /**
     * Placeholder items
     * @var array
     */
    protected $_items = array();

    /**
     * Set view
     *
     * @param  Zend_View_Interface $view
     * @return void
     */
    public function setView(Zend_View_Interface $view)
    {
        $this->view = $view;
    }

    /**
     * Placeholder helper
     *
     * @param  string $name
     * @return Zend_View_Helper_Placeholder_Container
     */
```

```
    public function placeholder($name)
    {
        $name = (string) $name;
        if (!isset($this->_items[$name])) {
            $this->_items[$name] = new Zend_View_Helper_Placeholder_Container(array());
        }
```

```
        return $this->_items[$name];
    }
}
```

Zend_View_Helper_HeadTitle

A small helper delegating to Zend_View_Helper_Placeholder which allows the user to utilise a default Placeholder for the express purpose of defining a page title for insertion into a Layout head element.

```
/**
 * Helper to insert or append <title> tags to the ZEND_HEAD Placeholder
 */
class Zend_View_Helper_HeadTitle
{
    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null

    /**
     * The default Zend_View_Helper_Placeholder instance
     *
     * @var Zend_View_Helper_Placeholder
     */
    protected $_placeholder = null;

    /**
     * Constants
     */
    const HEADTITLE_NAMESPACE = 'ZEND_HEAD_TITLE';

    /**
     * Constructor; assigns a Zend_View_Helper_Placeholder object.
     */
    public function __construct()
    {}

    /**
     * Return self for further in-object calls
     *
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function headTitle($value = null)
    {}

    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_HeadTitle
     */
    public function setView(Zend_View_Interface $view)
    {}
}
```

Zend_View_Helper_HeadMeta

A small helper delegating to Zend_View_Helper_Placeholder which allows the user to utilise a default Placeholder for the express purpose of defining page <meta> tags for insertion into a Layout head element.

```
/**
 * Helper to add a <meta> tag value to a head->meta Placeholder
 *
 * @category   Zend
 * @package    Zend_View
 * @subpackage Helper
 * @copyright  Copyright (c) 2007 Pádraic Brady
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 */
class Zend_View_Helper_HeadMeta
{

    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null;

    /**
     * The default Zend_View_Helper_Placeholder instance
     *
     * @var Zend_View_Helper_Placeholder
     */
    protected $_placeholder = null;

    /**
     * Constants
     */
    const HEADMETA_NAMESPACE = 'ZEND_HEAD_META';

    /**
     * Append a Head <meta> value if a parameter and
     * return self for further in-object calls
     *
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function headMeta($value = null)
    {}

    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_HeadTitle
     */
    public function setView(Zend_View_Interface $view)
    {}
}
```

Zend_View_Helper_HeadStyle

A small helper delegating to Zend_View_Helper_Placeholder which allows the user to utilise a default Placeholder for the express purpose of defining page <style> tags for insertion into a Layout head element.

```
/**
 * Helper to add a <style> tag value to a head->style Placeholder
 */
class Zend_View_Helper_HeadStyle
{

    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null;

    /**
     * The default Zend_View_Helper_Placeholder instance
     *
     * @var Zend_View_Helper_Placeholder
     */
    protected $_placeholder = null;

    /**
     * Constants
     */
    const HEADSTYLE_NAMESPACE = 'ZEND_HEAD_STYLE';

    /**
     * Append a Head <meta> value if a parameter and
     * return self for further in-object calls
     *
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function headStyle($value = null)
    {}

    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_HeadStyle
     */
    public function setView(Zend_View_Interface $view)
    {}
}
```

Zend_View_Helper_HeadScript

Include a Script file, or code block with a <head> element. Can avail of indexation to ensure Scripts are included in a preferred order.

```
/**
 * Helper to insert or append <script> tags to the head->script Placeholder
 */
class Zend_View_Helper_HeadScript
{

    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null;

    /**
     * The default Zend_View_Helper_Placeholder instance
     *
     * @var Zend_View_Helper_Placeholder
     */
    protected $_placeholder = null;

    /**
     * Constants
     */
    const HEADSCRIPT_NAMESPACE = 'ZEND_HEAD_SCRIPT';

    /**
     * Return self for further in-object calls
     *
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function headScript($file = null, $type = null, $index = null)
    {}
    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_HeadScript
     */
    public function setView(Zend_View_Interface $view)
    {}
}
```

Zend_View_Helper_Link

Add a Link element based on an array of attributes to be added to the <head> section of a template or layout. Although this can also handle style sheets, it may prove more useful to use the HeadStyle View Helper to segregate styles from other types of relationship links.

```
/**
 * Helper to add a <link> tag value to a head->link Placeholder
 */
class Zend_View_Helper_HeadLink
{
    /**
     * Instance of parent Zend_View object
     *
     * @var Zend_View_Abstract
     */
    public $view = null;

    /**
     * The default Zend_View_Helper_Placeholder instance
     *
     * @var Zend_View_Helper_Placeholder
     */
    protected $_placeholder = null;

    /**
     * Attributes common to <link> elements
     *
     * @var string
     */
    protected $_attributes = array('href', 'charset', 'hreflang', 'media', 'rel', 'rev', 'target',
'type', 'id', 'class', 'title', 'style', 'dir', 'lang', 'xml:lang');

    /**
     * Constants
     */
    const HEADLINK_NAMESPACE = 'ZEND_HEAD_LINK';

    /**
     * Set the Head <link> value if a parameter and
     * return self for further in-object call
     *
     * @param array $attributes
     * @return Zend_View_Helper_Placeholder_Container
     */
    public function headLink(array $attributes = null)
    {}

    /**
     * Set view object
     *
     * @param  Zend_View_Interface $view
     * @return Zend_View_Helper_HeadLink
     */
    public function setView(Zend_View_Interface $view)
    {}
}
```

Zend_View_Helper_Doctype

Obtain the Doctype string based on the Standard name for HTML, XHTML, MathML and SVG.

```
/**
 * Helper to obtain a doctype declaration based on the $standard parameter
 * which reflects the full name of the standard including version and context
 */
class Zend_View_Helper_Doctype
{

    public function doctype($standard = 'XHTML 1.0 Transitional')
    {}

    protected function _doctypeXhtml($standard)
    {}

    protected function _doctypeHtml($standard)
    {}

    protected function _doctypeMath($standard)
    {}

    protected function _doctypeSvg($standard)
    {}

}
```

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