Zend_Controller_Plugin_Template Proposal .. HAS TO BE REMOVED

<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_Controller_Plugin Component Proposal

Proposed Component Name	Zend_Controller_Plugin
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Controller_Plugin
Proposers	Aleksey V. Zapparov
Revision	1.1 - 14 January 2008: Initial release of proposal 1.2 - 21 May 2008: Ready for review (wiki revision: 10)

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
File: project/html/index.php
File: project/application/controllers/IndexController.php
File: project/application/templates/default.phtml
File: project/application/templates/printer.phtml
File: project/html/index.php
File: project/application/controllers/IndexController.php
File: project/application/controllers/AdminController.php
9. Class Skeletons

1. Overview

This plugin will help to use web templates.

2. References

- Web Template

3. Component Requirements, Constraints, and Acceptance Criteria

4. Dependencies on Other Framework Components

- Zend_Controller_Plugin_Abstract

5. Theory of Operation

This plugin can use simple or complex template name format.

- Simple format is used when you don't need different representations within single template.
- Complex format used when you want to make, for example, different templates with one design. For example it can be used to create different templates for admin panel and other part of web site but in one common design.

The use of Plugin very simple. All you have to do is - set path where your templates are stored (when initaiting new instance of object), then set template name's format and set default template name.

If you want to use Smarty with this plugin, all you have to do is extend Zend_View and setView($zend_view_smarty) where $zend_view_smarty is initiated object of Zend_View_Smarty, for more details on extending Zend_View see Zend Framework manual.

This plugin assigns all segments of $_body form Zend_Controller_Response to view's variables using their names. So if you'll appendBody() in your controller, it will be assigned to $this->default in view script.

6. Milestones / Tasks

! - TODO, ? - Idea

- !: Add availability to fall back to default template if specified template wasn't found.
- ?: [DONE] Rename plugin to Zend_Controller_Plugin_Template

7. Class Index

- Zend_Controller_Plugin_Template

8. Use Cases

UC-01 (Simple)

UC-02 (Complex)

9. Class Skeletons

```
class Zend_Controller_Plugin_Template extends Zend_Controller_Plugin_Abstract
```

```
{
    /**
     * Path where templates stored
     *
     * @var string
     */
    protected $_templateBasePath;

    /**
     * Name of default template which will be used if special segment ':template'
     * was not present in Response object.
     *
     * If you're using complex script name format {@link $_scriptNameFormat}
     * you'll have to use complex default template name and devide parts with
     * pipe '|'.
     *
     * @see $_scriptNameFormat
     * @var string
     */
    protected $_defaultTemplateName = 'default';

    /**
     * Format of script name composing. {name} will be replaced with template
     * name. So format can be '{name}/core.phtml' for complex templates.
     *
     * Complex script name format.
     * {name} also can be as {nameN}, where N=0, N+1. This might be useful for
     * more complex design templates. For example, you have two designs:
     * 'PinkFloyd' and 'VelvetUnderground'. And each of them has two templates:
     * 'admin' and 'default'. So you can specify $_scriptNameFormat as
     * '{name0}/{name1}.phtml'. But you'll also need to
     * {@link setDefaultTemplateName()} to 'PinkFloyd|default' if you want to use
     * 'PinkFloyd' design as default, and 'default' template as default.
     *
     * @see $_defaultTemplateName
     * @var string
     */
    protected $_scriptNameFormat = '{name}.phtml';

    /**
     * Instance of Zend_View.
     *
     * @var object
     */
    protected $_view;

    /**
     * Class constructor. Set path to where templates stored.
     *
     * @param string $pathToTemplates
     */
    public function __construct($pathToTemplates)
    {
        $this->_templateBasePath = (string) $pathToTemplates;
        if (!file_exists($this->_templateBasePath) || !is_dir($this->_templateBasePath)) {
            throw new Zend_Controller_Exception('Missing templates directory.');
        }
    }

    /**
     * Set default template's name.
     *
     * @see $_defaultTemplateName
     * @param string $name
     * @return Zend_Controller_Plugin_ResponsedView
     */
    public function setDefaultTemplateName($name)
    {
        $this->_defaultTemplateName = (string) $name;
        return $this;
```

```
    }

    /**
     * Retrieve default template's name.
     *
     * @see $_defaultTemplateName
     * @return string
     */
    public function getDefaultTemplateName()
    {
        return $this->_defaultTemplateName;
    }

    /**
     * Set format of script name composing.
     *
     * @see $_scriptNameFormat
     * @param string $format
     * @return Zend_Controller_Plugin_ResponsedView
     */
    public function setScriptNameFormat($format)
    {
        $this->_scriptNameFormat = (string) $format;
        return $this;
    }

    /**
     * Retrieve template script name.
     *
     * @param string $templateName
     * @return string
     */
    public function getScriptName($templateName = null)
    {
        $template_parts = explode('|', $this->getDefaultTemplateName());
        foreach (explode('|', (string) $templateName) as $key => $val) {
            if (null != $val) {
                $template_parts[$key] = $val;
            }
        }

        $script_name  = str_replace('{name}', $template_parts[0], $this->_scriptNameFormat);
        foreach ($template_parts as $id => $name) {
            $script_name = str_replace('{name' . $id . '}', $name, $script_name);
        }
        return $script_name;
    }

    /**
     * Set initiated view object (Subclass of {@link Zend_View}.
     *
     * @param object|null $view
     * @return Zend_Controller_Plugin_ResponsedView
     */
    public function setView($view = null)
    {
        if (null === $view) {
            $view = new Zend_View();
        }

        if (!($view instanceof Zend_View)) {
            throw new Zend_Controller_Exception('Invalid View object.');
        }

        $this->_view = $view;

        return $this;
    }

    /**
```

```
     * Get initiated view object.
     *
     * @see $_view
     * @see setView()
     * @return object
     */
    public function getView()
    {
        if (!isset($this->_view)) {
            $this->setView();
        }

        return $this->_view;
    }

    public function dispatchLoopShutdown()
    {
        $response = $this->getResponse();
        $template = $this->getScriptName($response->getBody(':template'));
        $response ->clearBody(':template');

        $view     = $this->getView();
        $view     ->setScriptPath($this->_templateBasePath);

        foreach ($response->getBody(true) as $segment => $content) {
            $view->assign($segment, $content);
        }

        $response ->clearBody();
        $response ->setBody($view->render($template));
    }
}
```

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