Zend_Http_Request - Sylvain Philip

<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_http_Request Component Proposal

Proposed Component Name	Zend_http_Request
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_http_Request
Proposers	Sylvain to contact me
Revision	1.1 - 1 August 2006: Updated from community comments. (wiki revision: 27)

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

Allow to personalize routing strategy in a request container

2. References

3. Component Requirements, Constraints, and Acceptance Criteria

- Correctly reads a developers mind for intent
- Generates strict PHP 5.x object oriented code codifying the read intent
- Performs with less than a 0.2 second lag from though to start of execution

4. Dependencies on Other Framework Components

- Zend_Controller_Dispatcher_Token

5. Theory of Operation

The component should be instantiated once in index.php and should be accessible from everywhere in ZF

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Controller_Dispatcher_Token
- Zend_http_Request

8. Use Cases

flexible capabilities to route

9. Class Skeletons

```
/*a request container proposition*/
interface Zend_http_Request
{
protected $modeRewrite = true; //rewriting or not
protected $Request = null; //current request
protected $Requests = array(); //array of requests
protected $URLbase = ''; //base url (http://www.mysite.com/test/)
//patern strategy for URi
protected $URIpaternOnRewrite ='[controller]/[action]/[pname]/[pvalue]';
protected $URIpaternNoRewrite ='[var]=[controller]_[action]&[pname]=[pvalue]';

/**
* Add a request (Zend_Controller_Dispatcher_Token object) in $Requests array
* @param string $requestName
* @param string $controllerName
* @param string $actionName
* @param array $params
* @return void
*/
public function addRequest($requestName, $controllerName, $actionName, $params=array());

/**
* set params to a request
* @param string $requestName
* @param array $params
* @return void
*/
public function setRequestParams($requestName, $paramsArray);

/**
* Get specific request
* @param string $requestName
* @return Zend_Controller_Dispatcher_Token object
*/
public function getRequest($requestName);

/**
* construct end send clean URL
* @param string $requestName
* @return string
*/
public function getRequestURL($requestName);

/**
* Get the current request.
* @return Zend_Controller_Dispatcher_Token object
*/
public function getCurrentRequest();

/**
* Get the current clean request URL.
* @return Zend_Controller_Dispatcher_Token object
*/
public function getCurrentRequestUrl();

/**
* Set mode rewriting or not
* @param bool $bool
*/
public function setModRewrite ($bool);

/**
* Set URI pattern to personalize routing strategy
* @param string $patern
*/
public function setUriPatern ($patern);

}
?>
```

Here is an class with code filled in. It is based on my 7 points below, a Martel's code and comments. I think this code shows more clearly how this class is proposed to work.

```
class Zend_Http_Request
{
protected $method;
protected $request;
protected $post;
protected $get;
protected $baseURL = '';
protected $pathInfo = '';
protected $controllerKey = 'controller';
protected $actionKey = 'action';
protected $params = array();

public function __construct($strict = false) {
    if ($strict) {
        $this->post = $_POST;
        $this->get = $_GET;
        unset($_POST);
        unset($_GET);
    } else {
        $this->post =& $_POST;
        $this->get =& $_GET;
    }

    $this->method = strtoupper($_SERVER['REQUEST_METHOD']);
    if ($this->method == 'POST') {
        $this->request =& $this->post;
    } else {
        $this->request =& $this->get;
    }
}


/* magic accessors to $_GET or $_POST depending on the request method */
public function __get($name) {
    return $this->request[$name];
}

public function __set($name, $value) {
    $this->request[$name] = $value;
}

public function __isset($name) {
    return isset($this->request[$name]);
}

public function has($name) {
    return __isset($name);
}

/* traditional accessors for stringy access -- these just call __get/__set */
public function get($name, $default = null) {
    if (! isset($this->request[$name])) {
        return $default;
    }
    return $this->__get($name);
}

public function set($name, $value) {
    $this->__set($name, $value);
}

/* direct accessors to $_GET or $_POST values */
```

```
public function getGET($name, $default = null) {
    if (! isset($this->get[$name])) {
        return $default;
    }
    return $this->get[$name];
}

public function setGET($name, $value) {
    $this->get[$name] = $value;
}

public function getPOST($name, $default = null) {
    if (! isset($this->post[$name])) {
        return $default;
    }
    return $this->post[$name];
}

public function setPOST($name, $value) {
    $this->post[$name] = $value;
}

/* is requset method true/false */
public function isPost() {
    return $this->method == 'POST';
}

/* alternate accessor to requset method 'GET'/'POST' */
public function getMethod() {
    return $this->method;
}

/* accessors for URL excluding PATH_INFO, setting would override value from system */
public function getBaseURL() {
    if (! $this->baseURL) {
        $this->baseURL = rtrim(dirname($_SERVER['SCRIPT_NAME']), '/');
    }
    return $this->baseURL;
}

public function setBaseURL($url) {
    $this->baseURL = $url;
}

/* accessors for PATH_INFO, setting would override value from system */
public function getPathInfo() {
    if (! $this->pathInfo) {
        $path = $_SERVER['REQUEST_URI'];
        if (strstr($path, '?')) {
            $path = substr($path, 0, strpos($path, '?'));
        }
        $url = $this->getBaseURL();
        if (strlen($url) > 0 && strpos($path, $url) === 0) {
            $path = substr($path, strlen($url));
        }
        $this->pathInfo = trim($path, '/');
    }
    return $this->pathInfo;
}

public function setPathInfo($value) {
    $this->pathInfo = $value;
}

/* accessors for controller parameter value */
public function getControllerName() {
    if (isset($this->get[$this->controllerKey])) {
        return $this->get[$this->controllerKey];
    } elseif (isset($this->post[$this->controllerKey])) {
        return $this->post[$this->controllerKey];
```

```
    }
}

public function setControllerName($controllerName) {
    $this->get[$this->controllerKey] = $controllerName;
}

/* accessor to set the controller parameter */
public function setControllerKey($key) {
    $this->controllerKey = $key;
}

/* accessors for action parameter value */
public function getActionName() {
    if (isset($this->get[$this->actionKey])) {
        return $this->get[$this->actionKey];
    } elseif (isset($this->post[$this->actionKey])) {
        return $this->post[$this->actionKey];
    }
}

public function setActionName($actionName) {
    $this->get[$this->actionKey] = $actionName;
}

/* accessor to set the controller parameter */
public function setActionKey($key) {
    $this->actionKey = $key;
}

/* accessors for user defined parameter array */
public function getParams() {
    return $this->params;
}

public function setParams($params) {
    $this->params = $params;
}

/* accessors for individual user defined parameters -- optional */
public function getParam($name) {
    return $this->params[$name];
}

public function setParam($name, $value) {
    $this->params[$name] = $value;
```

```
}

}
```

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

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