Zend_Stopwatch - Matthew Ratzloff

<ac:macro ac:name="info"><ac:parameter ac:name="title">New Proposal Template</ac:parameter><ac:rich-text-body>
This page has been created from a template that uses "zones." To proceed:

Edit the page
Replace sample content within each zone-data tag
Remove this notice
Save the page
When you are ready for review, remove the <ac:emoticon ac:name="warning" /> Under Construction notice

</ac:rich-text-body></ac:macro>

<ac:macro ac:name="note"><ac:parameter ac:name="title">Under Construction</ac:parameter><ac:rich-text-body>
This proposal is under construction and is not ready for review. </ac:rich-text-body></ac:macro>

<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_Stopwatch Component Proposal

Proposed Component Name	Zend_Stopwatch
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Stopwatch
Proposers	Matthew Ratzloff
Revision	0.1 - 10 April 2007: Just throwing up a page to get the idea out there. (wiki revision: 6)

Table of Contents

1. Overview

Zend_Stopwatch is a very simple class for timing events. It can be used for time-sensitive operations, benchmarking the speed of components, etc. At one point or another everyone needs something like this.

2. References

http://www.php.net/microtime

3. Component Requirements, Constraints, and Acceptance Criteria

Most requirements take the form of "foo will do ...." or "foo will not support ...", although different words and sentence structure might be used. Adding functionality to your proposal is requirements creep (bad), unless listed below. Discuss major changes with your team first, and then open a "feature improvement" issue against this component.

- This component **will** correctly reads a developers mind for intent and generate the right configuration file.
- The generated config file **will not** support XML, but will provide an extension point in the API.
- This component **will** use no more memory than twice the size of all data it contains.
- This component **will** include a factory method.
- This component **will not** allow subclassing. (i.e. when reviewed, we expect to see "final" keyword in code)
- This component **will** only generate data exports strictly complying with RFC 12345.
- This component **will** validate input data against formats supported by ZF component Foo.
- This component **will not** save any data using Zend_Cache or the filesystem. All transient data **will be** saved using Zend_Session.

4. Dependencies on Other Framework Components

- Zend_Exception

5. Theory of Operation

The component is instantiated with a mind-link that ...

6. Milestones / Tasks

Describe some intermediate state of this component in terms of design notes, additional material added to this page, and / code. Note any significant dependencies here, such as, "Milestone #3 can not be completed until feature Foo has been added to ZF component XYZ." Milestones will be required for acceptance of future proposals. They are not hard, and many times you will only need to think of the first three below.

- Milestone 1: design notes will be published here
- Milestone 2: Working prototype checked into the incubator supporting use cases #1, #2, ...
- Milestone 3: Working prototype checked into the incubator supporting use cases #3 and #4.
- Milestone 4: Unit tests exist, work, and are checked into SVN.
- Milestone 5: Initial documentation exists.

If a milestone is already done, begin the description with "[DONE]", like this:

- Milestone #: [DONE] Unit tests ...

7. Class Index

- Zend_Stopwatch

8. Use Cases

UC-01

... (see good use cases book)

9. Class Skeletons

```
<?php
/**
 * Determines execution time.
 */
class Zend_Stopwatch
{
    /**
     * Start time
     *
     * @var integer
     */
    protected $_startTime = 0;

    /**
     * Stop time
     *
     * @var integer
     */
    protected $_stopTime = 0;


    /**
     * Start the stopwatch.
     */
    public function start()
    {
        $this->_startTime += $this->_getTime() - $this->_stopTime;
        $this->_stopTime   = 0;
    }

    /**
     * Stops the stopwatch and return the elapsed time.
     *
     * @return float
     */
    public function stop()
    {
        $this->_stopTime = $this->_getTime();

        return $this->getElapsedSeconds();
    }

    /**
     * Resets the stopwatch.
     *
     * @return Zend_Stopwatch
     */
    public function reset()
```

```
{
    $this->_startTime = 0;
    $this->_stopTime  = 0;

    return $this;
}

/**
 * Is the stopwatch running?
 *
 * @return boolean
 */
public function isRunning()
{
    return ($this->_startTime != 0 and $this->_stopTime == 0);
}

/**
 * Returns the elapsed time, in seconds.
 *
 * @return float
 */
public function getElapsedSeconds()
{
    return $this->_getTime() - $this->_startTime;
}

/**
 * Returns the elapsed time, in milliseconds.
 *
 * @return float
 */
public function getElapsedMilliseconds()
{
    return $this->getElapsedSeconds() * 1000;
}

/**
 * Returns the elapsed time, in minutes.
 *
 * @return float
 */
public function getElapsedMinutes()
{
    return $this->getElapsedSeconds() / 60;
}

/**
 * Returns the elapsed time as a string.
 *
 * @return string
 */
public function __toString()
{
    return (string) $this->getElapsedSeconds();
}

/**
 * Returns the current Unix timestamp, in seconds.
```

```
     *
     * @return float
     */
    protected function _getTime()
    {
        return microtime(true);
```

```
        }
    }
```

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