ParallelRegex.php

```php
<?php
/**
 * Lexer adapted from Simple Test: http://sourceforge.net/projects/simpletest/
 * For an intro to the Lexer see:
 * https://web.archive.org/web/20120125041816/http://www.phppatterns.com/docs/develop/simple_test_lexer_notes
 *
 * @author Marcus Baker http://www.lastcraft.com
 */

namespace dokuwiki\Parsing\Lexer;

/**
 * Compounded regular expression.
 *
 * Any of the contained patterns could match and when one does it's label is returned.
 */
class ParallelRegex
{
    /** @var string[] patterns to match */
    protected $patterns;
    /** @var string[] labels for above patterns */
    protected $labels;
    /** @var string the compound regex matching all patterns */
    protected $regex;
    /** @var bool case sensitive matching? */
    protected $case;

    /**
     * Constructor. Starts with no patterns.
     *
     * @param boolean $case    True for case sensitive, false
     *                         for insensitive.
     */
    public function __construct($case)
    {
        $this->case = $case;
        $this->patterns = array();
        $this->labels = array();
        $this->regex = null;
    }

    /**
     * Adds a pattern with an optional label.
     *
```

```
45.      * @param mixed       $pattern Perl style regex. Must be UTF-8
46.      *                             encoded. If its a string, the
    (, )
47.      *                             lose their meaning unless they
48.      *                             form part of a lookahead or
49.      *                             lookbehind assertation.
50.      * @param bool|string $label   Label of regex to be returned
51.      *                             on a match. Label must be ASCII
52.      */
53.     public function addPattern($pattern, $label = true)
54.     {
55.         $count = count($this->patterns);
56.         $this->patterns[$count] = $pattern;
57.         $this->labels[$count] = $label;
58.         $this->regex = null;
59.     }
60.
61.     /**
62.      * Attempts to match all patterns at once against a string.
63.      *
64.      * @param string $subject      String to match against.
65.      * @param string $match        First matched portion of
66.      *                             subject.
67.      * @return bool|string         False if no match found, label
    if label exists, true if not
68.      */
69.     public function apply($subject, &$match)
70.     {
71.         if (count($this->patterns) == 0) {
72.             return false;
73.         }
74.         if (! preg_match($this->getCompoundedRegex(), $subject,
    $matches)) {
75.             $match = "";
76.             return false;
77.         }
78.
79.         $match = $matches[0];
80.         $size = count($matches);
81.         // FIXME this could be made faster by storing the labels
    as keys in a hashmap
82.         for ($i = 1; $i < $size; $i++) {
83.             if ($matches[$i] && isset($this->labels[$i - 1])) {
84.                 return $this->labels[$i - 1];
85.             }
86.         }
87.         return true;
88.     }
89.
```

```
90.     /**
91.      * Attempts to split the string against all patterns at once
92.      *
93.      * @param string $subject      String to match against.
94.      * @param array $split         The split result: array
    containing, pre-match, match & post-match strings
95.      * @return boolean             True on success.
96.      *
97.      * @author Christopher Smith <chris@jalakai.co.uk>
98.      */
99.     public function split($subject, &$split)
100.     {
101.         if (count($this->patterns) == 0) {
102.             return false;
103.         }
104. 
105.         if (! preg_match($this->getCompoundedRegex(), $subject,
    $matches)) {
106.             if (function_exists('preg_last_error')) {
107.                 $err = preg_last_error();
108.                 switch ($err) {
109.                     case PREG_BACKTRACK_LIMIT_ERROR:
110.                         msg('A PCRE backtrack error occured. Try
    to increase the pcre.backtrack_limit in php.ini', -1);
111.                         break;
112.                     case PREG_RECURSION_LIMIT_ERROR:
113.                         msg('A PCRE recursion error occured. Try
    to increase the pcre.recursion_limit in php.ini', -1);
114.                         break;
115.                     case PREG_BAD_UTF8_ERROR:
116.                         msg('A PCRE UTF-8 error occured. This
    might be caused by a faulty plugin', -1);
117.                         break;
118.                     case PREG_INTERNAL_ERROR:
119.                         msg('A PCRE internal error occured. This
    might be caused by a faulty plugin', -1);
120.                         break;
121.                 }
122.             }
123. 
124.             $split = array($subject, "", "");
125.             return false;
126.         }
127. 
128.         $idx = count($matches)-2;
129.         list($pre, $post) =
    preg_split($this->patterns[$idx].$this->getPerlMatchingFlags(),
    $subject, 2);
130.         $split = array($pre, $matches[0], $post);
131. 
132.         return isset($this->labels[$idx]) ? $this->labels[$idx] :
```

```
    true;
133.     }
134.
135.     /**
136.      * Compounds the patterns into a single
137.      * regular expression separated with the
138.      * "or" operator. Caches the regex.
139.      * Will automatically escape (, ) and / tokens.
140.      *
141.      * @return null|string
142.      */
143.     protected function getCompoundedRegex()
144.     {
145.         if ($this->regex == null) {
146.             $cnt = count($this->patterns);
147.             for ($i = 0; $i < $cnt; $i++) {
148.                 /*
149.                  * decompose the input pattern into "(", "(?",
    ")",
150.                  * "[...]", "[]..]", "[^]..]", "[...[:...:]..]",
    "\x"...
151.                  * elements.
152.                  */
153.                 preg_match_all('/\\\\.|' .
154.                                '\(\?|' .
155.                                '[()]|' .
156.
    '\[\^?\]?(?:\\\\.|\[:[^]]*:\]|[^]\\\\])*\]|' .
157.                                '[^[()\\\\]+/',
    $this->patterns[$i], $elts);
158.
159.                 $pattern = "";
160.                 $level = 0;
161.
162.                 foreach ($elts[0] as $elt) {
163.                     /*
164.                      * for "(", ")" remember the nesting level,
    add "\"
165.                      * only to the non-"(?" ones.
166.                      */
167.
168.                     switch ($elt) {
169.                         case '(':
170.                             $pattern .= '\(';
171.                             break;
172.                         case ')':
173.                             if ($level > 0)
174.                                 $level--; /* closing (? */
175.                             else $pattern .= '\\';
```

```
176.                             $pattern .= ')';
177.                             break;
178.                         case '(?':
179.                             $level++;
180.                             $pattern .= '(?';
181.                             break;
182.                         default:
183.                             if (substr($elt, 0, 1) == '\\')
184.                                 $pattern .= $elt;
185.                             else $pattern .= str_replace('/',
    '\/', $elt);
186.                         }
187.                 }
188.                 $this->patterns[$i] = "($pattern)";
189.             }
190.             $this->regex = "/" . implode("|", $this->patterns) .
    "/" . $this->getPerlMatchingFlags();
191.         }
192.         return $this->regex;
193.     }
194.
195.     /**
196.      * Accessor for perl regex mode flags to use.
197.      * @return string       Perl regex flags.
198.      */
199.     protected function getPerlMatchingFlags()
200.     {
201.         return ($this->case ? "msS" : "msSi");
202.     }
203. }
```