

Lexer.php

```
1. <?php
2. /**
3.  * Lexer adapted from Simple Test:
4.  * http://sourceforge.net/projects/simpletest/
5.  * For an intro to the Lexer see:
6.  *
7.  * https://web.archive.org/web/20120125041816/http://www.phppatterns.com/docs/develop/simple\_test\_lexer\_notes
8.  */
9.
10. namespace dokuwiki\Parsing\Lexer;
11.
12. /**
13.  * Accepts text and breaks it into tokens.
14.  *
15.  * Some optimisation to make the sure the content is only scanned
16.  * by the PHP regex
17.  * parser once. Lexer modes must not start with leading
18.  * underscores.
19.  */
20. class Lexer
21. {
22.     /** @var ParallelRegex[] */
23.     protected $regexes;
24.     /** @var \Doku_Handler */
25.     protected $handler;
26.     /** @var StateStack */
27.     protected $modeStack;
28.     /** @var array mode "rewrites" */
29.     protected $mode_handlers;
30.     /** @var bool case sensitive? */
31.     protected $case;
32.
33.     /**
34.      * Sets up the lexer in case insensitive matching by default.
35.      *
36.      * @param \Doku_Handler $handler Handling strategy by
37.      * reference.
38.      * @param string $start Starting handler.
39.      * @param boolean $case True for case sensitive.
40.      */
41.     public function __construct($handler, $start = "accept", $case = false)
42.     {
43.         $this->case = $case;
44.         $this->regexes = array();
```

```
42.         $this->handler = $handler;
43.         $this->modeStack = new StateStack($start);
44.         $this->mode_handlers = array();
45.     }
46.
47.     /**
48.      * Adds a token search pattern for a particular parsing mode.
49.      *
50.      * The pattern does not change the current mode.
51.      *
52.      * @param string $pattern      Perl style regex, but ( and )
53.      *                               lose the usual meaning.
54.      * @param string $mode          Should only apply this
55.      *                               pattern when dealing with
56.      *                               this type of input.
57.      */
58.     public function addPattern($pattern, $mode = "accept")
59.     {
60.         if (! isset($this->regexes[$mode])) {
61.             $this->regexes[$mode] = new
ParallelRegex($this->case);
62.         }
63.         $this->regexes[$mode]->addPattern($pattern);
64.     }
65.
66.     /**
67.      * Adds a pattern that will enter a new parsing mode.
68.      *
69.      * Useful for entering parenthesis, strings, tags, etc.
70.      *
71.      * @param string $pattern      Perl style regex, but ( and )
lose the usual meaning.
72.      * @param string $mode          Should only apply this pattern
when dealing with this type of input.
73.      * @param string $new_mode      Change parsing to this new
nested mode.
74.      */
75.     public function addEntryPattern($pattern, $mode, $new_mode)
76.     {
77.         if (! isset($this->regexes[$mode])) {
78.             $this->regexes[$mode] = new
ParallelRegex($this->case);
79.         }
80.         $this->regexes[$mode]->addPattern($pattern, $new_mode);
81.     }
82.
83.     /**
84.      * Adds a pattern that will exit the current mode and re-enter
the previous one.
85.      *
```

```
86.      * @param string $pattern      Perl style regex, but ( and )
      lose the usual meaning.
87.      * @param string $mode        Mode to leave.
88.      */
89.      public function addExitPattern($pattern, $mode)
90.      {
91.          if (! isset($this->regexes[$mode])) {
92.              $this->regexes[$mode] = new
ParallelRegex($this->case);
93.          }
94.          $this->regexes[$mode]->addPattern($pattern, "__exit");
95.      }
96.
97.      /**
98.       * Adds a pattern that has a special mode.
99.       *
100.      * Acts as an entry and exit pattern in one go, effectively
calling a special
101.      * parser handler for this token only.
102.      *
103.      * @param string $pattern      Perl style regex, but ( and )
lose the usual meaning.
104.      * @param string $mode        Should only apply this pattern
when dealing with this type of input.
105.      * @param string $special      Use this mode for this one
token.
106.      */
107.      public function addSpecialPattern($pattern, $mode, $special)
108.      {
109.          if (! isset($this->regexes[$mode])) {
110.              $this->regexes[$mode] = new
ParallelRegex($this->case);
111.          }
112.          $this->regexes[$mode]->addPattern($pattern, "__$special");
113.      }
114.
115.      /**
116.       * Adds a mapping from a mode to another handler.
117.       *
118.       * @param string $mode        Mode to be remapped.
119.       * @param string $handler      New target handler.
120.       */
121.      public function mapHandler($mode, $handler)
122.      {
123.          $this->mode_handlers[$mode] = $handler;
124.      }
125.
126.      /**
127.       * Splits the page text into tokens.
128.       *
129.       * Will fail if the handlers report an error or if no content
```

```
is consumed. If successful then each
130.     * unparsed and parsed token invokes a call to the held
    listener.
131.     *
132.     * @param string $raw      Raw HTML text.
133.     * @return boolean        True on success, else false.
134.     */
135.     public function parse($raw)
136.     {
137.         if (! isset($this->handler)) {
138.             return false;
139.         }
140.         $initialLength = strlen($raw);
141.         $length = $initialLength;
142.         $pos = 0;
143.         while (is_array($parsed = $this->reduce($raw))) {
144.             list($unmatched, $matched, $mode) = $parsed;
145.             $currentLength = strlen($raw);
146.             $matchPos = $initialLength - $currentLength -
strlen($matched);
147.             if (!$this->dispatchTokens($unmatched, $matched,
$mode, $pos, $matchPos)) {
148.                 return false;
149.             }
150.             if ($currentLength == $length) {
151.                 return false;
152.             }
153.             $length = $currentLength;
154.             $pos = $initialLength - $currentLength;
155.         }
156.         if (!$parsed) {
157.             return false;
158.         }
159.         return $this->invokeHandler($raw, DOKU_LEXER_UNMATCHED,
$pos);
160.     }
161.
162.     /**
163.     * Gives plugins access to the mode stack
164.     *
165.     * @return StateStack
166.     */
167.     public function getModeStack()
168.     {
169.         return $this->modeStack;
170.     }
171.
172.     /**
173.     * Sends the matched token and any leading unmatched
174.     * text to the parser changing the lexer to a new
```

```
175.      * mode if one is listed.
176.      *
177.      * @param string $unmatched Unmatched leading portion.
178.      * @param string $matched Actual token match.
179.      * @param bool|string $mode Mode after match. A boolean false
mode causes no change.
180.      * @param int $initialPos
181.      * @param int $matchPos Current byte index location in raw doc
thats being parsed
182.      * @return boolean          False if there was any error
from the parser.
183.      */
184.      protected function dispatchTokens($unmatched, $matched, $mode,
$initialPos, $matchPos)
185.      {
186.          if (! $this->invokeHandler($unmatched,
DOKU_LEXER_UNMATCHED, $initialPos)) {
187.              return false;
188.          }
189.          if ($this->isModeEnd($mode)) {
190.              if (! $this->invokeHandler($matched, DOKU_LEXER_EXIT,
$matchPos)) {
191.                  return false;
192.              }
193.              return $this->modeStack->leave();
194.          }
195.          if ($this->isSpecialMode($mode)) {
196.              $this->modeStack->enter($this->decodeSpecial($mode));
197.              if (! $this->invokeHandler($matched,
DOKU_LEXER_SPECIAL, $matchPos)) {
198.                  return false;
199.              }
200.              return $this->modeStack->leave();
201.          }
202.          if (is_string($mode)) {
203.              $this->modeStack->enter($mode);
204.              return $this->invokeHandler($matched,
DOKU_LEXER_ENTER, $matchPos);
205.          }
206.          return $this->invokeHandler($matched, DOKU_LEXER_MATCHED,
$matchPos);
207.      }
208.
209.      /**
210.       * Tests to see if the new mode is actually to leave the
current mode and pop an item from the matching
211.       * mode stack.
212.       *
213.       * @param string $mode      Mode to test.
214.       * @return boolean          True if this is the exit mode.
215.       */
```

```
216.     protected function isModeEnd($mode)
217.     {
218.         return ($mode === "__exit");
219.     }
220.
221.     /**
222.      * Test to see if the mode is one where this mode is entered
223.      * for this token only and automatically
224.      * leaves immediately afterwards.
225.      *
226.      * @param string $mode      Mode to test.
227.      * @return boolean          True if this is the exit mode.
228.      */
229.     protected function isSpecialMode($mode)
230.     {
231.         return (strcmp($mode, "_", 1) == 0);
232.     }
233.
234.     /**
235.      * Strips the magic underscore marking single token modes.
236.      *
237.      * @param string $mode      Mode to decode.
238.      * @return string          Underlying mode name.
239.      */
240.     protected function decodeSpecial($mode)
241.     {
242.         return substr($mode, 1);
243.     }
244.
245.     /**
246.      * Calls the parser method named after the current mode.
247.      *
248.      * Empty content will be ignored. The lexer has a parser
249.      * handler for each mode in the lexer.
250.      *
251.      * @param string $content Text parsed.
252.      * @param boolean $is_match Token is recognised rather
253.      * than unparsed data.
254.      * @param int $pos Current byte index location in raw doc
255.      * thats being parsed
256.      * @return bool
257.      */
258.     protected function invokeHandler($content, $is_match, $pos)
259.     {
260.         if (($content === "") || ($content === false)) {
261.             return true;
262.         }
263.         $handler = $this->modeStack->getCurrent();
264.         if (isset($this->mode_handlers[$handler])) {
265.             $handler = $this->mode_handlers[$handler];
```

```
264.     }
265.
266.     // modes starting with plugin_ are all handled by the same
267.     // handler but with an additional parameter
268.     if (substr($handler, 0, 7)=='plugin_') {
269.         list($handler,$plugin) = sexplode('_', $handler, 2,
270.         '');
271.         return $this->handler->$handler($content, $is_match,
272.         $pos, $plugin);
273.     }
274.     return $this->handler->$handler($content, $is_match,
275.     $pos);
276.     /**
277.      * Tries to match a chunk of text and if successful removes
278.      * the recognised chunk and any leading
279.      * unparsed data. Empty strings will not be matched.
280.      *
281.      * @param string $raw          The subject to parse. This is
282.      * the content that will be eaten.
283.      * @return array|bool          Three item list of unparsed
284.      * content followed by the
285.      * recognised token and finally the
286.      * action the parser is to take.
287.      *
288.      * True if no match, false if there
289.      * is a parsing error.
290.      */
291.     protected function reduce(&$raw)
292.     {
293.         if (!
294.         isset($this->regexes[$this->modeStack->getCurrent()])) {
295.             return false;
296.         }
297.         if ($raw === "") {
298.             return true;
299.         }
300.         if ($action =
301.         $this->regexes[$this->modeStack->getCurrent()]->split($raw,
302.         $split)) {
303.             list($unparsed, $match, $raw) = $split;
304.             return array($unparsed, $match, $action);
305.         }
306.         return true;
307.     }
308.     /**
309.      * Escapes regex characters other than (, ) and /
310.      *
311.      * @param string $str
```

```
304.      * @return string
305.      */
306.      public static function escape($str)
307.      {
308.          $chars = array(
309.              '\\\\\\\\',
310.              '\\.',
311.              '\\+',
312.              '\\*',
313.              '\\?',
314.              '\\[',
315.              '\\^',
316.              '\\]',
317.              '\\$',
318.              '\\{',
319.              '\\}',
320.              '\\=',
321.              '\\!',
322.              '\\<',
323.              '\\>',
324.              '\\|',
325.              '\\:/'
326.          );
327.
328.          $escaped = array(
329.              '\\\\\\\\\\\\',
330.              '\\.',
331.              '\\+',
332.              '\\*',
333.              '\\?',
334.              '\\[',
335.              '\\^',
336.              '\\]',
337.              '\\$',
338.              '\\{',
339.              '\\}',
340.              '\\=',
341.              '\\!',
342.              '\\<',
343.              '\\>',
344.              '\\|',
345.              '\\:'
346.          );
347.          return preg_replace($chars, $escaped, $str);
348.      }
349. }
```

From:
<https://wwoss.ru/> - **worldwide open-source software**

Permanent link:
<https://wwoss.ru/doku.php?id=wiki:xref:dokuwiki:inc:parsing:lexer:lexer.php>

Last update: **2025/01/16 18:55**

