Lexer.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;

/**
 * Accepts text and breaks it into tokens.
 *
 * Some optimisation to make the sure the content is only scanned by the PHP regex
 * parser once. Lexer modes must not start with leading underscores.
 */
class Lexer
{
    /** @var ParallelRegex[] */
    protected $regexes;
    /** @var \Doku_Handler */
    protected $handler;
    /** @var StateStack */
    protected $modeStack;
    /** @var array mode "rewrites" */
    protected $mode_handlers;
    /** @var bool case sensitive? */
    protected $case;

    /**
     * Sets up the lexer in case insensitive matching by default.
     *
     * @param \Doku_Handler $handler  Handling strategy by reference.
     * @param string $start           Starting handler.
     * @param boolean $case           True for case sensitive.
     */
    public function __construct($handler, $start = "accept", $case = false)
    {
        $this->case = $case;
        $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
    for this token only and automatically
223.      * leaves immediately afterwoods.
224.      *
225.      * @param string $mode    Mode to test.
226.      * @return boolean        True if this is the exit mode.
227.      */
228.     protected function isSpecialMode($mode)
229.     {
230.         return (strncmp($mode, "_", 1) == 0);
231.     }
232.
233.     /**
234.      * Strips the magic underscore marking single token modes.
235.      *
236.      * @param string $mode    Mode to decode.
237.      * @return string         Underlying mode name.
238.      */
239.     protected function decodeSpecial($mode)
240.     {
241.         return substr($mode, 1);
242.     }
243.
244.     /**
245.      * Calls the parser method named after the current mode.
246.      *
247.      * Empty content will be ignored. The lexer has a parser
    handler for each mode in the lexer.
248.      *
249.      * @param string $content Text parsed.
250.      * @param boolean $is_match Token is recognised rather
251.      *                                  than unparsed data.
252.      * @param int $pos Current byte index location in raw doc
253.      *                                thats being parsed
254.      * @return bool
255.      */
256.     protected function invokeHandler($content, $is_match, $pos)
257.     {
258.         if (($content === "") || ($content === false)) {
259.             return true;
260.         }
```

```
261.         $handler = $this->modeStack->getCurrent();
262.         if (isset($this->mode_handlers[$handler])) {
263.             $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.             return $this->handler->$handler($content, $is_match,
    $pos, $plugin);
271.         }
272.
273.         return $this->handler->$handler($content, $is_match,
    $pos);
274.     }
275.
276.     /**
277.      * Tries to match a chunk of text and if successful removes
    the recognised chunk and any leading
278.      * unparsed data. Empty strings will not be matched.
279.      *
280.      * @param string $raw         The subject to parse. This is
    the content that will be eaten.
281.      * @return array|bool         Three item list of unparsed
    content followed by the
282.      *                            recognised token and finally the
    action the parser is to take.
283.      *                            True if no match, false if there
    is a parsing error.
284.      */
285.     protected function reduce(&$raw)
286.     {
287.         if (!
    isset($this->regexes[$this->modeStack->getCurrent()])) {
288.             return false;
289.         }
290.         if ($raw === "") {
291.             return true;
292.         }
293.         if ($action =
    $this->regexes[$this->modeStack->getCurrent()]->split($raw,
    $split)) {
294.             list($unparsed, $match, $raw) = $split;
295.             return array($unparsed, $match, $action);
296.         }
297.         return true;
298.     }
299.
300.     /**
```

```
301.      * Escapes regex characters other than (, ) and /
302.      *
303.      * @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. }
```