

Lexer.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();
        $this->handler = $handler;
        $this->modeStack = new StateStack($start);
        $this->mode_handlers = array();
    }
}
```

```
}

/**
 * Adds a token search pattern for a particular parsing mode.
 *
 * The pattern does not change the current mode.
 *
 * @param string $pattern      Perl style regex, but ( and )
 *                             lose the usual meaning.
 * @param string $mode         Should only apply this
 *                             pattern when dealing with
 *                             this type of input.
 */
public function addPattern($pattern, $mode = "accept")
{
    if (! isset($this->regexes[$mode])) {
        $this->regexes[$mode] = new ParallelRegex($this->case);
    }
    $this->regexes[$mode]->addPattern($pattern);
}

/**
 * Adds a pattern that will enter a new parsing mode.
 *
 * Useful for entering parenthesis, strings, tags, etc.
 *
 * @param string $pattern      Perl style regex, but ( and ) lose
the usual meaning.
 * @param string $mode         Should only apply this pattern when
dealing with this type of input.
 * @param string $new_mode     Change parsing to this new nested
mode.
 */
public function addEntryPattern($pattern, $mode, $new_mode)
{
    if (! isset($this->regexes[$mode])) {
        $this->regexes[$mode] = new ParallelRegex($this->case);
    }
    $this->regexes[$mode]->addPattern($pattern, $new_mode);
}

/**
 * Adds a pattern that will exit the current mode and re-enter the
previous one.
 *
 * @param string $pattern      Perl style regex, but ( and ) lose
the usual meaning.
 * @param string $mode         Mode to leave.
 */
```

```
public function addExitPattern($pattern, $mode)
{
    if (! isset($this->regexes[$mode])) {
        $this->regexes[$mode] = new ParallelRegex($this->case);
    }
    $this->regexes[$mode]->addPattern($pattern, "__exit");
}

/**
 * Adds a pattern that has a special mode.
 *
 * Acts as an entry and exit pattern in one go, effectively calling
 a special
 * parser handler for this token only.
 *
 * @param string $pattern      Perl style regex, but ( and ) lose
 the usual meaning.
 * @param string $mode        Should only apply this pattern when
 dealing with this type of input.
 * @param string $special     Use this mode for this one token.
 */
public function addSpecialPattern($pattern, $mode, $special)
{
    if (! isset($this->regexes[$mode])) {
        $this->regexes[$mode] = new ParallelRegex($this->case);
    }
    $this->regexes[$mode]->addPattern($pattern, "__$special");
}

/**
 * Adds a mapping from a mode to another handler.
 *
 * @param string $mode        Mode to be remapped.
 * @param string $handler     New target handler.
 */
public function mapHandler($mode, $handler)
{
    $this->mode_handlers[$mode] = $handler;
}

/**
 * Splits the page text into tokens.
 *
 * Will fail if the handlers report an error or if no content is
 consumed. If successful then each
 * unparsed and parsed token invokes a call to the held listener.
 *
 * @param string $raw        Raw HTML text.
 * @return boolean          True on success, else false.
 */
public function parse($raw)
```

```
{
    if (! isset($this->handler)) {
        return false;
    }
    $initialLength = strlen($raw);
    $length = $initialLength;
    $pos = 0;
    while (is_array($parsed = $this->reduce($raw))) {
        list($unmatched, $matched, $mode) = $parsed;
        $currentLength = strlen($raw);
        $matchPos = $initialLength - $currentLength -
strlen($matched);
        if (! $this->dispatchTokens($unmatched, $matched, $mode,
$pos, $matchPos)) {
            return false;
        }
        if ($currentLength == $length) {
            return false;
        }
        $length = $currentLength;
        $pos = $initialLength - $currentLength;
    }
    if (!$parsed) {
        return false;
    }
    return $this->invokeHandler($raw, DOKU_LEXER_UNMATCHED, $pos);
}

/**
 * Gives plugins access to the mode stack
 *
 * @return StateStack
 */
public function getModeStack()
{
    return $this->modeStack;
}

/**
 * Sends the matched token and any leading unmatched
 * text to the parser changing the lexer to a new
 * mode if one is listed.
 *
 * @param string $unmatched Unmatched leading portion.
 * @param string $matched Actual token match.
 * @param bool|string $mode Mode after match. A boolean false mode
causes no change.
 * @param int $initialPos
 * @param int $matchPos Current byte index location in raw doc
```

```

thats being parsed
    * @return boolean          False if there was any error from
the parser.
    */
    protected function dispatchTokens($unmatched, $matched, $mode,
$initialPos, $matchPos)
    {
        if (! $this->invokeHandler($unmatched, DOKU_LEXER_UNMATCHED,
$initialPos)) {
            return false;
        }
        if ($this->isModeEnd($mode)) {
            if (! $this->invokeHandler($matched, DOKU_LEXER_EXIT,
$matchPos)) {
                return false;
            }
            return $this->modeStack->leave();
        }
        if ($this->isSpecialMode($mode)) {
            $this->modeStack->enter($this->decodeSpecial($mode));
            if (! $this->invokeHandler($matched, DOKU_LEXER_SPECIAL,
$matchPos)) {
                return false;
            }
            return $this->modeStack->leave();
        }
        if (is_string($mode)) {
            $this->modeStack->enter($mode);
            return $this->invokeHandler($matched, DOKU_LEXER_ENTER,
$matchPos);
        }
        return $this->invokeHandler($matched, DOKU_LEXER_MATCHED,
$matchPos);
    }

    /**
     * Tests to see if the new mode is actually to leave the current
     mode and pop an item from the matching
     * mode stack.
     *
     * @param string $mode    Mode to test.
     * @return boolean        True if this is the exit mode.
     */
    protected function isModeEnd($mode)
    {
        return ($mode === "__exit");
    }

    /**
     * Test to see if the mode is one where this mode is entered for
     this token only and automatically

```

```
* leaves immediately afterwoods.
*
* @param string $mode      Mode to test.
* @return boolean          True if this is the exit mode.
*/
protected function isSpecialMode($mode)
{
    return (strcmp($mode, "_", 1) == 0);
}

/**
* Strips the magic underscore marking single token modes.
*
* @param string $mode      Mode to decode.
* @return string          Underlying mode name.
*/
protected function decodeSpecial($mode)
{
    return substr($mode, 1);
}

/**
* Calls the parser method named after the current mode.
*
* Empty content will be ignored. The lexer has a parser handler
for each mode in the lexer.
*
* @param string $content Text parsed.
* @param boolean $is_match Token is recognised rather
                           than unparsed data.
* @param int $pos Current byte index location in raw doc
                           thats being parsed
* @return bool
*/
protected function invokeHandler($content, $is_match, $pos)
{
    if (($content === "") || ($content === false)) {
        return true;
    }
    $handler = $this->modeStack->getCurrent();
    if (isset($this->mode_handlers[$handler])) {
        $handler = $this->mode_handlers[$handler];
    }

    // modes starting with plugin_ are all handled by the same
    // handler but with an additional parameter
    if (substr($handler, 0, 7)=='plugin_') {
        list($handler,$plugin) = sexplode('_', $handler, 2, '');
        return $this->handler->$handler($content, $is_match, $pos,
```

```

$plugin);
    }

    return $this->handler->$handler($content, $is_match, $pos);
}

/**
 * Tries to match a chunk of text and if successful removes the
 * recognised chunk and any leading
 * unparsed data. Empty strings will not be matched.
 *
 * @param string $raw          The subject to parse. This is the
 * content that will be eaten.
 * @return array|bool          Three item list of unparsed content
 * followed by the
 * recognised token and finally the
 * action the parser is to take.
 *
 * True if no match, false if there is a
 * parsing error.
 */
protected function reduce(&$raw)
{
    if (! isset($this->regexes[$this->modeStack->getCurrent()])) {
        return false;
    }
    if ($raw === "") {
        return true;
    }
    if ($action =
$this->regexes[$this->modeStack->getCurrent()]->split($raw, $split)) {
        list($unparsed, $match, $raw) = $split;
        return array($unparsed, $match, $action);
    }
    return true;
}

/**
 * Escapes regex characters other than (, ) and /
 *
 * @param string $str
 * @return string
 */
public static function escape($str)
{
    $chars = array(
        '/\\\\\\',
        '/\\./',
        '/\\+/',
        '/\\*/',
        '/\\?/',
        '/\\[/',

```

```
        '\^/' ,
        '\]/' ,
        '\$/' ,
        '\{' ,
        '\}' ,
        '\=' ,
        '\!/' ,
        '\</' ,
        '\>/' ,
        '\|/' ,
        '\:/'
    );

    $escaped = array(
        '\\\\\\\\',
        '\.' ,
        '\+' ,
        '\*' ,
        '\?' ,
        '\[' ,
        '\^' ,
        '\]' ,
        '\$' ,
        '\{' ,
        '\}' ,
        '\=' ,
        '\!' ,
        '\<' ,
        '\>' ,
        '\|' ,
        '\:'
    );
    return preg_replace($chars, $escaped, $str);
}
```

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

Permanent link:
<https://wwoss.ru/doku.php?id=wiki:xref:dokuwiki:inc:parsing:lexer:lexer.php&rev=1737023034>

Last update: 2025/01/16 13:23

