

Block.php

```
1. <?php
2.
3. namespace dokuwiki\Parsing\Handler;
4.
5. /**
6.  * Handler for paragraphs
7.  *
8.  * @author Harry Fuecks <hfuecks@gmail.com>
9.  */
10. class Block
11. {
12.     protected $calls = array();
13.     protected $skipEol = false;
14.     protected $inParagraph = false;
15.
16.     // Blocks these should not be inside paragraphs
17.     protected $blockOpen = array(
18.         'header',
19.
20.         'listu_open', 'listo_open', 'listitem_open', 'listcontent_open',
21.
22.         'table_open', 'tablerow_open', 'tablecell_open', 'tableheader_open', '
23.         tablethead_open',
24.         'quote_open',
25.         'code', 'file', 'hr', 'preformatted', 'rss',
26.         'footnote_open',
27.     );
28.
29.     protected $blockClose = array(
30.         'header',
31.
32.         'listu_close', 'listo_close', 'listitem_close', 'listcontent_close',
33.
34.         'table_close', 'tablerow_close', 'tablecell_close', 'tableheader_clos
35.         e', 'tablethead_close',
36.         'quote_close',
37.         'code', 'file', 'hr', 'preformatted', 'rss',
38.         'footnote_close',
39.     );
40.
41.     // Stacks can contain paragraphs
42.     protected $stackOpen = array(
43.         'section_open',
44.     );
45.
46.     protected $stackClose = array(
47.         'section_close',
48.     );
49. }
```

```
43.
44.
45.     /**
46.      * Constructor. Adds loaded syntax plugins to the block and
47.      * stack
48.      * arrays
49.      *
50.      * @author Andreas Gohr <andi@splitbrain.org>
51.      */
52.     public function __construct()
53.     {
54.         global $DOKU_PLUGINS;
55.         //check if syntax plugins were loaded
56.         if (empty($DOKU_PLUGINS['syntax'])) return;
57.         foreach ($DOKU_PLUGINS['syntax'] as $n => $p) {
58.             $ptype = $p->getPType();
59.             if ($ptype == 'block') {
60.                 $this->blockOpen[] = 'plugin_'. $n;
61.                 $this->blockClose[] = 'plugin_'. $n;
62.             } elseif ($ptype == 'stack') {
63.                 $this->stackOpen[] = 'plugin_'. $n;
64.                 $this->stackClose[] = 'plugin_'. $n;
65.             }
66.         }
67.
68.         protected function openParagraph($pos)
69.         {
70.             if ($this->inParagraph) return;
71.             $this->calls[] = array('p_open', array(), $pos);
72.             $this->inParagraph = true;
73.             $this->skipEol = true;
74.         }
75.
76.         /**
77.          * Close a paragraph if needed
78.          *
79.          * This function makes sure there are no empty paragraphs on
80.          * the stack
81.          *
82.          * @author Andreas Gohr <andi@splitbrain.org>
83.          *
84.          * @param string|integer $pos
85.          */
86.         protected function closeParagraph($pos)
87.         {
88.             if (!$this->inParagraph) return;
89.             // look back if there was any content - we don't want
90.             empty paragraphs
```

```

89.         $content = '';
90.         $ccount = count($this->calls);
91.         for ($i=$ccount-1; $i>=0; $i--) {
92.             if ($this->calls[$i][0] == 'p_open') {
93.                 break;
94.             } elseif ($this->calls[$i][0] == 'cdata') {
95.                 $content .= $this->calls[$i][1][0];
96.             } else {
97.                 $content = 'found markup';
98.                 break;
99.             }
100.        }
101.
102.        if (trim($content)=='') {
103.            //remove the whole paragraph
104.            //array_splice($this->calls,$i); // <- this is much
            slower than the loop below
105.            for ($x=$ccount; $x>$i;
106.                $x--) array_pop($this->calls);
107.        } else {
108.            // remove ending linebreaks in the paragraph
109.            $i=count($this->calls)-1;
110.            if ($this->calls[$i][0] == 'cdata')
111.                $this->calls[$i][1][0] = rtrim($this->calls[$i][1][0], "\n");
112.            $this->calls[] = array('p_close',array(), $pos);
113.        }
114.        $this->inParagraph = false;
115.        $this->skipEol = true;
116.    }
117.
118.    protected function addCall($call)
119.    {
120.        $key = count($this->calls);
121.        if ($key and ($call[0] == 'cdata') and
122.            ($this->calls[$key-1][0] == 'cdata')) {
123.            $this->calls[$key-1][1][0] .= $call[1][0];
124.        } else {
125.            $this->calls[] = $call;
126.        }
127.
128.        // simple version of addCall, without checking cdata
129.        protected function storeCall($call)
130.        {
131.            $this->calls[] = $call;
132.        }
133.
134.        /**
135.         * Processes the whole instruction stack to open and close
            paragraphs

```

```
136.      *
137.      * @author Harry Fuecks <hfuecks@gmail.com>
138.      * @author Andreas Gohr <andi@splitbrain.org>
139.      *
140.      * @param array $calls
141.      *
142.      * @return array
143.      */
144.      public function process($calls)
145.      {
146.          // open first paragraph
147.          $this->openParagraph(0);
148.          foreach ($calls as $key => $call) {
149.              $cname = $call[0];
150.              if ($cname == 'plugin') {
151.                  $cname='plugin_'. $call[1][0];
152.                  $plugin = true;
153.                  $plugin_open = (($call[1][2] == DOKU_LEXER_ENTER)
|| ($call[1][2] == DOKU_LEXER_SPECIAL));
154.                  $plugin_close = (($call[1][2] == DOKU_LEXER_EXIT)
|| ($call[1][2] == DOKU_LEXER_SPECIAL));
155.              } else {
156.                  $plugin = false;
157.              }
158.              /* stack */
159.              if (in_array($cname, $this->stackClose) && (!$plugin
|| $plugin_close)) {
160.                  $this->closeParagraph($call[2]);
161.                  $this->storeCall($call);
162.                  $this->openParagraph($call[2]);
163.                  continue;
164.              }
165.              if (in_array($cname, $this->stackOpen) && (!$plugin ||
$plugin_open)) {
166.                  $this->closeParagraph($call[2]);
167.                  $this->storeCall($call);
168.                  $this->openParagraph($call[2]);
169.                  continue;
170.              }
171.              /* block */
172.              // If it's a substitution it opens and closes at the
same call.
173.              // To make sure next paragraph is correctly started,
let close go first.
174.              if (in_array($cname, $this->blockClose) && (!$plugin
|| $plugin_close)) {
175.                  $this->closeParagraph($call[2]);
176.                  $this->storeCall($call);
177.                  $this->openParagraph($call[2]);
```

```
178.         continue;
179.     }
180.     if (in_array($name, $this->blockOpen) && (!$plugin ||
    $plugin_open)) {
181.         $this->closeParagraph($call[2]);
182.         $this->storeCall($call);
183.         continue;
184.     }
185.     /* eol */
186.     if ($name == 'eol') {
187.         // Check this isn't an eol instruction to skip...
188.         if (!$this->skipEol) {
189.             // Next is EOL => double eol => mark as
    paragraph
190.             if (isset($calls[$key+1]) && $calls[$key+1][0]
    == 'eol') {
191.                 $this->closeParagraph($call[2]);
192.                 $this->openParagraph($call[2]);
193.             } else {
194.                 //if this is just a single eol make a
    space from it
195.                 $this->addCall(array('cdata', array("\n"),
    $call[2]));
196.             }
197.         }
198.         continue;
199.     }
200.     /* normal */
201.     $this->addCall($call);
202.     $this->skipEol = false;
203. }
204. // close last paragraph
205. $call = end($this->calls);
206. $this->closeParagraph($call[2]);
207. return $this->calls;
208. }
209. }
```

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

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

Last update: **2025/01/16 22:02**

