handler.php

```php
<?php

use dokuwiki\Extension\Event;
use dokuwiki\Extension\SyntaxPlugin;
use dokuwiki\Parsing\Handler\Block;
use dokuwiki\Parsing\Handler\CallWriter;
use dokuwiki\Parsing\Handler\CallWriterInterface;
use dokuwiki\Parsing\Handler\Lists;
use dokuwiki\Parsing\Handler\Nest;
use dokuwiki\Parsing\Handler\Preformatted;
use dokuwiki\Parsing\Handler\Quote;
use dokuwiki\Parsing\Handler\Table;

/**
 * Class Doku_Handler
 */
class Doku_Handler {
    /** @var CallWriterInterface */
    protected $callWriter = null;

    /** @var array The current CallWriter will write directly to this list of calls, Parser reads it */
    public $calls = array();

    /** @var array internal status holders for some modes */
    protected $status = array(
        'section' => false,
        'doublequote' => 0,
    );

    /** @var bool should blocks be rewritten? FIXME seems to always be true */
    protected $rewriteBlocks = true;

    /**
     * @var bool are we in a footnote already?
     */
    protected $footnote;

    /**
     * Doku_Handler constructor.
     */
    public function __construct() {
        $this->callWriter = new CallWriter($this);
    }

    /**
     * Add a new call by passing it to the current CallWriter
```

```
47.      *
48.      * @param string $handler handler method name (see mode
    handlers below)
49.      * @param mixed $args arguments for this call
50.      * @param int $pos  byte position in the original source file
51.      */
52.     public function addCall($handler, $args, $pos) {
53.         $call = array($handler,$args, $pos);
54.         $this->callWriter->writeCall($call);
55.     }
56.
57.     /**
58.      * Accessor for the current CallWriter
59.      *
60.      * @return CallWriterInterface
61.      */
62.     public function getCallWriter() {
63.         return $this->callWriter;
64.     }
65.
66.     /**
67.      * Set a new CallWriter
68.      *
69.      * @param CallWriterInterface $callWriter
70.      */
71.     public function setCallWriter($callWriter) {
72.         $this->callWriter = $callWriter;
73.     }
74.
75.     /**
76.      * Return the current internal status of the given name
77.      *
78.      * @param string $status
79.      * @return mixed|null
80.      */
81.     public function getStatus($status) {
82.         if (!isset($this->status[$status])) return null;
83.         return $this->status[$status];
84.     }
85.
86.     /**
87.      * Set a new internal status
88.      *
89.      * @param string $status
90.      * @param mixed $value
91.      */
92.     public function setStatus($status, $value) {
93.         $this->status[$status] = $value;
94.     }
95.
```

```
96.     /** @deprecated 2019-10-31 use addCall() instead */
97.     public function _addCall($handler, $args, $pos) {
98.         dbg_deprecated('addCall');
99.         $this->addCall($handler, $args, $pos);
100.    }
101.
102.    /**
103.     * Similar to addCall, but adds a plugin call
104.     *
105.     * @param string $plugin name of the plugin
106.     * @param mixed $args arguments for this call
107.     * @param int $state a LEXER_STATE_* constant
108.     * @param int $pos byte position in the original source file
109.     * @param string $match matched syntax
110.     */
111.    public function addPluginCall($plugin, $args, $state, $pos,
    $match) {
112.        $call = array('plugin',array($plugin, $args, $state,
    $match), $pos);
113.        $this->callWriter->writeCall($call);
114.    }
115.
116.    /**
117.     * Finishes handling
118.     *
119.     * Called from the parser. Calls finalise() on the call
    writer, closes open
120.     * sections, rewrites blocks and adds document_start and
    document_end calls.
121.     *
122.     * @triggers PARSER_HANDLER_DONE
123.     */
124.    public function finalize(){
125.        $this->callWriter->finalise();
126.
127.        if ( $this->status['section'] ) {
128.            $last_call = end($this->calls);
129.            array_push($this->calls,array('section_close',array(),
    $last_call[2]));
130.        }
131.
132.        if ( $this->rewriteBlocks ) {
133.            $B = new Block();
134.            $this->calls = $B->process($this->calls);
135.        }
136.
137.        Event::createAndTrigger('PARSER_HANDLER_DONE',$this);
138.
139.
    array_unshift($this->calls,array('document_start',array(),0));
140.        $last_call = end($this->calls);
```

```
141.
    array_push($this->calls,array('document_end',array(),$last_call[2]
    ));
142.     }
143.
144.     /**
145.      * fetch the current call and advance the pointer to the next
    one
146.      *
147.      * @fixme seems to be unused?
148.      * @return bool|mixed
149.      */
150.     public function fetch() {
151.         $call = current($this->calls);
152.         if($call !== false) {
153.             next($this->calls); //advance the pointer
154.             return $call;
155.         }
156.         return false;
157.     }
158.
159.
160.     /**
161.      * Internal function for parsing highlight options.
162.      * $options is parsed for key value pairs separated by commas.
163.      * A value might also be missing in which case the value will
    simple
164.      * be set to true. Commas in strings are ignored, e.g.
    option="4,56"
165.      * will work as expected and will only create one entry.
166.      *
167.      * @param string $options space separated list of key-value
    pairs,
168.      *                        e.g. option1=123, option2="456"
169.      * @return array|null     Array of key-value pairs
    $array['key'] = 'value';
170.      *                        or null if no entries found
171.      */
172.     protected function parse_highlight_options($options) {
173.         $result = array();
174.
    preg_match_all('/(\w+(?:="[^"]*"))|(\w+(?:=[^\s]*))|(\w+[^=\s\]])(
    ?:\s*)/', $options, $matches, PREG_SET_ORDER);
175.         foreach ($matches as $match) {
176.             $equal_sign = strpos($match [0], '=');
177.             if ($equal_sign === false) {
178.                 $key = trim($match[0]);
179.                 $result [$key] = 1;
180.             } else {
181.                 $key = substr($match[0], 0, $equal_sign);
```

```
182.                 $value = substr($match[0], $equal_sign+1);
183.                 $value = trim($value, '"');
184.                 if (strlen($value) > 0) {
185.                     $result [$key] = $value;
186.                 } else {
187.                     $result [$key] = 1;
188.                 }
189.             }
190.         }
191.
192.         // Check for supported options
193.         $result = array_intersect_key(
194.             $result,
195.             array_flip(array(
196.                             'enable_line_numbers',
197.                             'start_line_numbers_at',
198.                             'highlight_lines_extra',
199.                             'enable_keyword_links')
200.             )
201.         );
202.
203.         // Sanitize values
204.         if(isset($result['enable_line_numbers'])) {
205.             if($result['enable_line_numbers'] === 'false') {
206.                 $result['enable_line_numbers'] = false;
207.             }
208.             $result['enable_line_numbers'] = (bool)
    $result['enable_line_numbers'];
209.         }
210.         if(isset($result['highlight_lines_extra'])) {
211.             $result['highlight_lines_extra'] = array_map('intval',
    explode(',', $result['highlight_lines_extra']));
212.             $result['highlight_lines_extra'] =
    array_filter($result['highlight_lines_extra']);
213.             $result['highlight_lines_extra'] =
    array_unique($result['highlight_lines_extra']);
214.         }
215.         if(isset($result['start_line_numbers_at'])) {
216.             $result['start_line_numbers_at'] = (int)
    $result['start_line_numbers_at'];
217.         }
218.         if(isset($result['enable_keyword_links'])) {
219.             if($result['enable_keyword_links'] === 'false') {
220.                 $result['enable_keyword_links'] = false;
221.             }
222.             $result['enable_keyword_links'] = (bool)
    $result['enable_keyword_links'];
223.         }
224.         if (count($result) == 0) {
225.             return null;
226.         }
```

```
227.
228.         return $result;
229.     }
230.
231.     /**
232.      * Simplifies handling for the formatting tags which all
    behave the same
233.      *
234.      * @param string $match matched syntax
235.      * @param int $state a LEXER_STATE_* constant
236.      * @param int $pos byte position in the original source file
237.      * @param string $name actual mode name
238.      */
239.     protected function nestingTag($match, $state, $pos, $name) {
240.         switch ( $state ) {
241.             case DOKU_LEXER_ENTER:
242.                 $this->addCall($name.'_open', array(), $pos);
243.                 break;
244.             case DOKU_LEXER_EXIT:
245.                 $this->addCall($name.'_close', array(), $pos);
246.                 break;
247.             case DOKU_LEXER_UNMATCHED:
248.                 $this->addCall('cdata', array($match), $pos);
249.                 break;
250.         }
251.     }
252.
253.
254.     /**
255.      * The following methods define the handlers for the different
    Syntax modes
256.      *
257.      * The handlers are called from
    dokuwiki\Parsing\Lexer\Lexer\invokeParser()
258.      *
259.      * @todo it might make sense to move these into their own
    class or merge them with the
260.      *       ParserMode classes some time.
261.      */
262.     // region mode handlers
263.
264.     /**
265.      * Special plugin handler
266.      *
267.      * This handler is called for all modes starting with
    'plugin_'.
268.      * An additional parameter with the plugin name is passed. The
    plugin's handle()
269.      * method is called here
270.      *
```

```
271.      * @author Andreas Gohr <andi@splitbrain.org>
272.      *
273.      * @param string $match matched syntax
274.      * @param int $state a LEXER_STATE_* constant
275.      * @param int $pos byte position in the original source file
276.      * @param string $pluginname name of the plugin
277.      * @return bool mode handled?
278.      */
279.     public function plugin($match, $state, $pos, $pluginname){
280.         $data = array($match);
281.         /** @var SyntaxPlugin $plugin */
282.         $plugin = plugin_load('syntax',$pluginname);
283.         if($plugin != null){
284.             $data = $plugin->handle($match, $state, $pos, $this);
285.         }
286.         if ($data !== false) {
287.
    $this->addPluginCall($pluginname,$data,$state,$pos,$match);
288.         }
289.         return true;
290.     }
291.
292.     /**
293.      * @param string $match matched syntax
294.      * @param int $state a LEXER_STATE_* constant
295.      * @param int $pos byte position in the original source file
296.      * @return bool mode handled?
297.      */
298.     public function base($match, $state, $pos) {
299.         switch ( $state ) {
300.             case DOKU_LEXER_UNMATCHED:
301.                 $this->addCall('cdata', array($match), $pos);
302.                 return true;
303.             break;
304.         }
305.         return false;
306.     }
307.
308.     /**
309.      * @param string $match matched syntax
310.      * @param int $state a LEXER_STATE_* constant
311.      * @param int $pos byte position in the original source file
312.      * @return bool mode handled?
313.      */
314.     public function header($match, $state, $pos) {
315.         // get level and title
316.         $title = trim($match);
317.         $level = 7 - strspn($title,'=');
318.         if($level < 1) $level = 1;
319.         $title = trim($title,'=');
320.         $title = trim($title);
```

```
321.
322.         if ($this->status['section'])
     $this->addCall('section_close', array(), $pos);
323.
324.         $this->addCall('header', array($title, $level, $pos),
     $pos);
325.
326.         $this->addCall('section_open', array($level), $pos);
327.         $this->status['section'] = true;
328.         return true;
329.     }
330.
331.     /**
332.      * @param string $match matched syntax
333.      * @param int $state a LEXER_STATE_* constant
334.      * @param int $pos byte position in the original source file
335.      * @return bool mode handled?
336.      */
337.     public function notoc($match, $state, $pos) {
338.         $this->addCall('notoc', array(), $pos);
339.         return true;
340.     }
341.
342.     /**
343.      * @param string $match matched syntax
344.      * @param int $state a LEXER_STATE_* constant
345.      * @param int $pos byte position in the original source file
346.      * @return bool mode handled?
347.      */
348.     public function nocache($match, $state, $pos) {
349.         $this->addCall('nocache', array(), $pos);
350.         return true;
351.     }
352.
353.     /**
354.      * @param string $match matched syntax
355.      * @param int $state a LEXER_STATE_* constant
356.      * @param int $pos byte position in the original source file
357.      * @return bool mode handled?
358.      */
359.     public function linebreak($match, $state, $pos) {
360.         $this->addCall('linebreak', array(), $pos);
361.         return true;
362.     }
363.
364.     /**
365.      * @param string $match matched syntax
366.      * @param int $state a LEXER_STATE_* constant
367.      * @param int $pos byte position in the original source file
368.      * @return bool mode handled?
```

```
369.      */
370.     public function eol($match, $state, $pos) {
371.         $this->addCall('eol', array(), $pos);
372.         return true;
373.     }
374.
375.     /**
376.      * @param string $match matched syntax
377.      * @param int $state a LEXER_STATE_* constant
378.      * @param int $pos byte position in the original source file
379.      * @return bool mode handled?
380.      */
381.     public function hr($match, $state, $pos) {
382.         $this->addCall('hr', array(), $pos);
383.         return true;
384.     }
385.
386.     /**
387.      * @param string $match matched syntax
388.      * @param int $state a LEXER_STATE_* constant
389.      * @param int $pos byte position in the original source file
390.      * @return bool mode handled?
391.      */
392.     public function strong($match, $state, $pos) {
393.         $this->nestingTag($match, $state, $pos, 'strong');
394.         return true;
395.     }
396.
397.     /**
398.      * @param string $match matched syntax
399.      * @param int $state a LEXER_STATE_* constant
400.      * @param int $pos byte position in the original source file
401.      * @return bool mode handled?
402.      */
403.     public function emphasis($match, $state, $pos) {
404.         $this->nestingTag($match, $state, $pos, 'emphasis');
405.         return true;
406.     }
407.
408.     /**
409.      * @param string $match matched syntax
410.      * @param int $state a LEXER_STATE_* constant
411.      * @param int $pos byte position in the original source file
412.      * @return bool mode handled?
413.      */
414.     public function underline($match, $state, $pos) {
415.         $this->nestingTag($match, $state, $pos, 'underline');
416.         return true;
417.     }
418.
419.     /**
```

```
420.      * @param string $match matched syntax
421.      * @param int $state a LEXER_STATE_* constant
422.      * @param int $pos byte position in the original source file
423.      * @return bool mode handled?
424.      */
425.     public function monospace($match, $state, $pos) {
426.         $this->nestingTag($match, $state, $pos, 'monospace');
427.         return true;
428.     }
429.
430.     /**
431.      * @param string $match matched syntax
432.      * @param int $state a LEXER_STATE_* constant
433.      * @param int $pos byte position in the original source file
434.      * @return bool mode handled?
435.      */
436.     public function subscript($match, $state, $pos) {
437.         $this->nestingTag($match, $state, $pos, 'subscript');
438.         return true;
439.     }
440.
441.     /**
442.      * @param string $match matched syntax
443.      * @param int $state a LEXER_STATE_* constant
444.      * @param int $pos byte position in the original source file
445.      * @return bool mode handled?
446.      */
447.     public function superscript($match, $state, $pos) {
448.         $this->nestingTag($match, $state, $pos, 'superscript');
449.         return true;
450.     }
451.
452.     /**
453.      * @param string $match matched syntax
454.      * @param int $state a LEXER_STATE_* constant
455.      * @param int $pos byte position in the original source file
456.      * @return bool mode handled?
457.      */
458.     public function deleted($match, $state, $pos) {
459.         $this->nestingTag($match, $state, $pos, 'deleted');
460.         return true;
461.     }
462.
463.     /**
464.      * @param string $match matched syntax
465.      * @param int $state a LEXER_STATE_* constant
466.      * @param int $pos byte position in the original source file
467.      * @return bool mode handled?
468.      */
469.     public function footnote($match, $state, $pos) {
```

```
470.         if (!isset($this->footnote)) $this->footnote = false;
471.
472.         switch ( $state ) {
473.             case DOKU_LEXER_ENTER:
474.                 // footnotes can not be nested - however due to
    limitations in lexer it can't be prevented
475.                 // we will still enter a new footnote mode, we
    just do nothing
476.                 if ($this->footnote) {
477.                     $this->addCall('cdata', array($match), $pos);
478.                     break;
479.                 }
480.                 $this->footnote = true;
481.
482.                 $this->callWriter = new Nest($this->callWriter,
    'footnote_close');
483.                 $this->addCall('footnote_open', array(), $pos);
484.             break;
485.             case DOKU_LEXER_EXIT:
486.                 // check whether we have already exitted the
    footnote mode, can happen if the modes were nested
487.                 if (!$this->footnote) {
488.                     $this->addCall('cdata', array($match), $pos);
489.                     break;
490.                 }
491.
492.                 $this->footnote = false;
493.                 $this->addCall('footnote_close', array(), $pos);
494.
495.                 /** @var Nest $reWriter */
496.                 $reWriter = $this->callWriter;
497.                 $this->callWriter = $reWriter->process();
498.             break;
499.             case DOKU_LEXER_UNMATCHED:
500.                 $this->addCall('cdata', array($match), $pos);
501.             break;
502.         }
503.         return true;
504.     }
505.
506.     /**
507.      * @param string $match matched syntax
508.      * @param int $state a LEXER_STATE_* constant
509.      * @param int $pos byte position in the original source file
510.      * @return bool mode handled?
511.      */
512.     public function listblock($match, $state, $pos) {
513.         switch ( $state ) {
514.             case DOKU_LEXER_ENTER:
515.                 $this->callWriter = new Lists($this->callWriter);
516.                 $this->addCall('list_open', array($match), $pos);
```

```
517.                 break;
518.                 case DOKU_LEXER_EXIT:
519.                     $this->addCall('list_close', array(), $pos);
520.                     /** @var Lists $reWriter */
521.                     $reWriter = $this->callWriter;
522.                     $this->callWriter = $reWriter->process();
523.                 break;
524.                 case DOKU_LEXER_MATCHED:
525.                     $this->addCall('list_item', array($match), $pos);
526.                 break;
527.                 case DOKU_LEXER_UNMATCHED:
528.                     $this->addCall('cdata', array($match), $pos);
529.                 break;
530.             }
531.             return true;
532.         }
533.
534.         /**
535.          * @param string $match matched syntax
536.          * @param int $state a LEXER_STATE_* constant
537.          * @param int $pos byte position in the original source file
538.          * @return bool mode handled?
539.          */
540.         public function unformatted($match, $state, $pos) {
541.             if ( $state == DOKU_LEXER_UNMATCHED ) {
542.                 $this->addCall('unformatted', array($match), $pos);
543.             }
544.             return true;
545.         }
546.
547.         /**
548.          * @param string $match matched syntax
549.          * @param int $state a LEXER_STATE_* constant
550.          * @param int $pos byte position in the original source file
551.          * @return bool mode handled?
552.          */
553.         public function preformatted($match, $state, $pos) {
554.             switch ( $state ) {
555.                 case DOKU_LEXER_ENTER:
556.                     $this->callWriter = new
    Preformatted($this->callWriter);
557.                     $this->addCall('preformatted_start', array(),
    $pos);
558.                 break;
559.                 case DOKU_LEXER_EXIT:
560.                     $this->addCall('preformatted_end', array(), $pos);
561.                     /** @var Preformatted $reWriter */
562.                     $reWriter = $this->callWriter;
563.                     $this->callWriter = $reWriter->process();
564.                 break;
```

```
565.             case DOKU_LEXER_MATCHED:
566.                 $this->addCall('preformatted_newline', array(),
    $pos);
567.             break;
568.             case DOKU_LEXER_UNMATCHED:
569.                 $this->addCall('preformatted_content',
    array($match), $pos);
570.             break;
571.         }
572. 
573.         return true;
574.     }
575. 
576.     /**
577.      * @param string $match matched syntax
578.      * @param int $state a LEXER_STATE_* constant
579.      * @param int $pos byte position in the original source file
580.      * @return bool mode handled?
581.      */
582.     public function quote($match, $state, $pos) {
583. 
584.         switch ( $state ) {
585. 
586.             case DOKU_LEXER_ENTER:
587.                 $this->callWriter = new Quote($this->callWriter);
588.                 $this->addCall('quote_start', array($match),
    $pos);
589.             break;
590. 
591.             case DOKU_LEXER_EXIT:
592.                 $this->addCall('quote_end', array(), $pos);
593.                 /** @var Lists $reWriter */
594.                 $reWriter = $this->callWriter;
595.                 $this->callWriter = $reWriter->process();
596.             break;
597. 
598.             case DOKU_LEXER_MATCHED:
599.                 $this->addCall('quote_newline', array($match),
    $pos);
600.             break;
601. 
602.             case DOKU_LEXER_UNMATCHED:
603.                 $this->addCall('cdata', array($match), $pos);
604.             break;
605. 
606.         }
607. 
608.         return true;
609.     }
610. 
611.     /**
```

```
612.      * @param string $match matched syntax
613.      * @param int $state a LEXER_STATE_* constant
614.      * @param int $pos byte position in the original source file
615.      * @return bool mode handled?
616.      */
617.     public function file($match, $state, $pos) {
618.         return $this->code($match, $state, $pos, 'file');
619.     }
620.
621.     /**
622.      * @param string $match matched syntax
623.      * @param int $state a LEXER_STATE_* constant
624.      * @param int $pos byte position in the original source file
625.      * @param string $type either 'code' or 'file'
626.      * @return bool mode handled?
627.      */
628.     public function code($match, $state, $pos, $type='code') {
629.         if ( $state == DOKU_LEXER_UNMATCHED ) {
630.             $matches = sexplode('>',$match,2,'');
631.             // Cut out variable options enclosed in []
632.             preg_match('/\[.*\]/', $matches[0], $options);
633.             if (!empty($options[0])) {
634.                 $matches[0] = str_replace($options[0], '',
    $matches[0]);
635.             }
636.             $param = preg_split('/\s+/', $matches[0], 2,
    PREG_SPLIT_NO_EMPTY);
637.             while(count($param) < 2) array_push($param, null);
638.             // We shortcut html here.
639.             if ($param[0] == 'html') $param[0] = 'html4strict';
640.             if ($param[0] == '-') $param[0] = null;
641.             array_unshift($param, $matches[1]);
642.             if (!empty($options[0])) {
643.                 $param [] = $this->parse_highlight_options
    ($options[0]);
644.             }
645.             $this->addCall($type, $param, $pos);
646.         }
647.         return true;
648.     }
649.
650.     /**
651.      * @param string $match matched syntax
652.      * @param int $state a LEXER_STATE_* constant
653.      * @param int $pos byte position in the original source file
654.      * @return bool mode handled?
655.      */
656.     public function acronym($match, $state, $pos) {
657.         $this->addCall('acronym', array($match), $pos);
658.         return true;
```

```
659.     }
660.
661.     /**
662.      * @param string $match matched syntax
663.      * @param int $state a LEXER_STATE_* constant
664.      * @param int $pos byte position in the original source file
665.      * @return bool mode handled?
666.      */
667.     public function smiley($match, $state, $pos) {
668.         $this->addCall('smiley', array($match), $pos);
669.         return true;
670.     }
671.
672.     /**
673.      * @param string $match matched syntax
674.      * @param int $state a LEXER_STATE_* constant
675.      * @param int $pos byte position in the original source file
676.      * @return bool mode handled?
677.      */
678.     public function wordblock($match, $state, $pos) {
679.         $this->addCall('wordblock', array($match), $pos);
680.         return true;
681.     }
682.
683.     /**
684.      * @param string $match matched syntax
685.      * @param int $state a LEXER_STATE_* constant
686.      * @param int $pos byte position in the original source file
687.      * @return bool mode handled?
688.      */
689.     public function entity($match, $state, $pos) {
690.         $this->addCall('entity', array($match), $pos);
691.         return true;
692.     }
693.
694.     /**
695.      * @param string $match matched syntax
696.      * @param int $state a LEXER_STATE_* constant
697.      * @param int $pos byte position in the original source file
698.      * @return bool mode handled?
699.      */
700.     public function multiplyentity($match, $state, $pos) {
701.         preg_match_all('/\d+/',$match,$matches);
702.         $this->addCall('multiplyentity', array($matches[0][0],
    $matches[0][1]), $pos);
703.         return true;
704.     }
705.
706.     /**
707.      * @param string $match matched syntax
708.      * @param int $state a LEXER_STATE_* constant
```

```
709.      * @param int $pos byte position in the original source file
710.      * @return bool mode handled?
711.      */
712.     public function singlequoteopening($match, $state, $pos) {
713.         $this->addCall('singlequoteopening', array(), $pos);
714.         return true;
715.     }
716.
717.     /**
718.      * @param string $match matched syntax
719.      * @param int $state a LEXER_STATE_* constant
720.      * @param int $pos byte position in the original source file
721.      * @return bool mode handled?
722.      */
723.     public function singlequoteclosing($match, $state, $pos) {
724.         $this->addCall('singlequoteclosing', array(), $pos);
725.         return true;
726.     }
727.
728.     /**
729.      * @param string $match matched syntax
730.      * @param int $state a LEXER_STATE_* constant
731.      * @param int $pos byte position in the original source file
732.      * @return bool mode handled?
733.      */
734.     public function apostrophe($match, $state, $pos) {
735.         $this->addCall('apostrophe', array(), $pos);
736.         return true;
737.     }
738.
739.     /**
740.      * @param string $match matched syntax
741.      * @param int $state a LEXER_STATE_* constant
742.      * @param int $pos byte position in the original source file
743.      * @return bool mode handled?
744.      */
745.     public function doublequoteopening($match, $state, $pos) {
746.         $this->addCall('doublequoteopening', array(), $pos);
747.         $this->status['doublequote']++;
748.         return true;
749.     }
750.
751.     /**
752.      * @param string $match matched syntax
753.      * @param int $state a LEXER_STATE_* constant
754.      * @param int $pos byte position in the original source file
755.      * @return bool mode handled?
756.      */
757.     public function doublequoteclosing($match, $state, $pos) {
758.         if ($this->status['doublequote'] <= 0) {
```

```
759.             $this->doublequoteopening($match, $state, $pos);
760.         } else {
761.             $this->addCall('doublequoteclosing', array(), $pos);
762.             $this->status['doublequote'] = max(0, --
    $this->status['doublequote']);
763.         }
764.         return true;
765.     }
766.
767.     /**
768.      * @param string $match matched syntax
769.      * @param int $state a LEXER_STATE_* constant
770.      * @param int $pos byte position in the original source file
771.      * @return bool mode handled?
772.      */
773.     public function camelcaselink($match, $state, $pos) {
774.         $this->addCall('camelcaselink', array($match), $pos);
775.         return true;
776.     }
777.
778.     /**
779.      * @param string $match matched syntax
780.      * @param int $state a LEXER_STATE_* constant
781.      * @param int $pos byte position in the original source file
782.      * @return bool mode handled?
783.      */
784.     public function internallink($match, $state, $pos) {
785.         // Strip the opening and closing markup
786.         $link =
    preg_replace(array('/^\[\[/','/\]\]$/u'),'',$match);
787.
788.         // Split title from URL
789.         $link = sexplode('|',$link,2);
790.         if ( $link[1] === null ) {
791.             $link[1] = null;
792.         } else if ( preg_match('/^\{\{[^\}]+\}\}$/',$link[1]) ) {
793.             // If the title is an image, convert it to an array
    containing the image details
794.             $link[1] = Doku_Handler_Parse_Media($link[1]);
795.         }
796.         $link[0] = trim($link[0]);
797.
798.         //decide which kind of link it is
799.
800.         if ( link_isinterwiki($link[0]) ) {
801.             // Interwiki
802.             $interwiki = sexplode('>',$link[0],2,'');
803.             $this->addCall(
804.                 'interwikilink',
805.
    array($link[0],$link[1],strtolower($interwiki[0]),$interwiki[1]),
```

```
806.                 $pos
807.                 );
808.         }elseif ( preg_match('/^\\\\\\\\[^\\\\]+?\\\\/u',$link[0])
     ) {
809.             // Windows Share
810.             $this->addCall(
811.                 'windowssharelink',
812.                 array($link[0],$link[1]),
813.                 $pos
814.                 );
815.         }elseif ( preg_match('#^([a-z0-9\-\.+]+?)://#i',$link[0])
     ) {
816.             // external link (accepts all protocols)
817.             $this->addCall(
818.                     'externallink',
819.                     array($link[0],$link[1]),
820.                     $pos
821.                     );
822.         }elseif (
     preg_match('<'.PREG_PATTERN_VALID_EMAIL.'>',$link[0]) ) {
823.             // E-Mail (pattern above is defined in inc/mail.php)
824.             $this->addCall(
825.                 'emaillink',
826.                 array($link[0],$link[1]),
827.                 $pos
828.                 );
829.         }elseif ( preg_match('!^#.+!',$link[0]) ){
830.             // local link
831.             $this->addCall(
832.                 'locallink',
833.                 array(substr($link[0],1),$link[1]),
834.                 $pos
835.                 );
836.         }else{
837.             // internal link
838.             $this->addCall(
839.                 'internallink',
840.                 array($link[0],$link[1]),
841.                 $pos
842.                 );
843.         }
844.
845.         return true;
846.     }
847.
848.     /**
849.      * @param string $match matched syntax
850.      * @param int $state a LEXER_STATE_* constant
851.      * @param int $pos byte position in the original source file
852.      * @return bool mode handled?
```

```
853.      */
854.     public function filelink($match, $state, $pos) {
855.         $this->addCall('filelink', array($match, null), $pos);
856.         return true;
857.     }
858.
859.     /**
860.      * @param string $match matched syntax
861.      * @param int $state a LEXER_STATE_* constant
862.      * @param int $pos byte position in the original source file
863.      * @return bool mode handled?
864.      */
865.     public function windowssharelink($match, $state, $pos) {
866.         $this->addCall('windowssharelink', array($match, null),
    $pos);
867.         return true;
868.     }
869.
870.     /**
871.      * @param string $match matched syntax
872.      * @param int $state a LEXER_STATE_* constant
873.      * @param int $pos byte position in the original source file
874.      * @return bool mode handled?
875.      */
876.     public function media($match, $state, $pos) {
877.         $p = Doku_Handler_Parse_Media($match);
878.
879.         $this->addCall(
880.               $p['type'],
881.               array($p['src'], $p['title'], $p['align'],
    $p['width'],
882.                     $p['height'], $p['cache'], $p['linking']),
883.               $pos
884.              );
885.         return true;
886.     }
887.
888.     /**
889.      * @param string $match matched syntax
890.      * @param int $state a LEXER_STATE_* constant
891.      * @param int $pos byte position in the original source file
892.      * @return bool mode handled?
893.      */
894.     public function rss($match, $state, $pos) {
895.         $link =
    preg_replace(array('/^\{\{rss>/','/\}\}$/'),'',$match);
896.
897.         // get params
898.         list($link, $params) = sexplode(' ', $link, 2, '');
899.
900.         $p = array();
```

```
901.         if(preg_match('/\b(\d+)\b/',$params,$match)){
902.             $p['max'] = $match[1];
903.         }else{
904.             $p['max'] = 8;
905.         }
906.         $p['reverse'] = (preg_match('/rev/',$params));
907.         $p['author']  = (preg_match('/\b(by|author)/',$params));
908.         $p['date']    = (preg_match('/\b(date)/',$params));
909.         $p['details'] = (preg_match('/\b(desc|detail)/',$params));
910.         $p['nosort']  = (preg_match('/\b(nosort)\b/',$params));
911.
912.         if (preg_match('/\b(\d+)([dhm])\b/',$params,$match)) {
913.             $period = array('d' => 86400, 'h' => 3600, 'm' => 60);
914.             $p['refresh'] = max(600,$match[1]*$period[$match[2]]);
    // n * period in seconds, minimum 10 minutes
915.         } else {
916.             $p['refresh'] = 14400;   // default to 4 hours
917.         }
918.
919.         $this->addCall('rss', array($link, $p), $pos);
920.         return true;
921.     }
922.
923.     /**
924.      * @param string $match matched syntax
925.      * @param int $state a LEXER_STATE_* constant
926.      * @param int $pos byte position in the original source file
927.      * @return bool mode handled?
928.      */
929.     public function externallink($match, $state, $pos) {
930.         $url   = $match;
931.         $title = null;
932.
933.         // add protocol on simple short URLs
934.         if(substr($url,0,3) == 'ftp' && (substr($url,0,6) !=
    'ftp://')){
935.             $title = $url;
936.             $url   = 'ftp://'.$url;
937.         }
938.         if(substr($url,0,3) == 'www' && (substr($url,0,7) !=
    'http://')){
939.             $title = $url;
940.             $url = 'http://'.$url;
941.         }
942.
943.         $this->addCall('externallink', array($url, $title), $pos);
944.         return true;
945.     }
946.
947.     /**
```

```
948.      * @param string $match matched syntax
949.      * @param int $state a LEXER_STATE_* constant
950.      * @param int $pos byte position in the original source file
951.      * @return bool mode handled?
952.      */
953.     public function emaillink($match, $state, $pos) {
954.         $email = preg_replace(array('/^</','/>$/'),'',$match);
955.         $this->addCall('emaillink', array($email, null), $pos);
956.         return true;
957.     }
958.
959.     /**
960.      * @param string $match matched syntax
961.      * @param int $state a LEXER_STATE_* constant
962.      * @param int $pos byte position in the original source file
963.      * @return bool mode handled?
964.      */
965.     public function table($match, $state, $pos) {
966.         switch ( $state ) {
967.
968.             case DOKU_LEXER_ENTER:
969.
970.                 $this->callWriter = new Table($this->callWriter);
971.
972.                 $this->addCall('table_start', array($pos + 1),
    $pos);
973.                 if ( trim($match) == '^' ) {
974.                     $this->addCall('tableheader', array(), $pos);
975.                 } else {
976.                     $this->addCall('tablecell', array(), $pos);
977.                 }
978.             break;
979.
980.             case DOKU_LEXER_EXIT:
981.                 $this->addCall('table_end', array($pos), $pos);
982.                 /** @var Table $reWriter */
983.                 $reWriter = $this->callWriter;
984.                 $this->callWriter = $reWriter->process();
985.             break;
986.
987.             case DOKU_LEXER_UNMATCHED:
988.                 if ( trim($match) != '' ) {
989.                     $this->addCall('cdata', array($match), $pos);
990.                 }
991.             break;
992.
993.             case DOKU_LEXER_MATCHED:
994.                 if ( $match == ' ' ){
995.                     $this->addCall('cdata', array($match), $pos);
996.                 } else if ( preg_match('/:::/',$match) ) {
997.                     $this->addCall('rowspan', array($match),
```

```
    $pos);
998.                 } else if ( preg_match('/\t+/',$match) ) {
999.                     $this->addCall('table_align', array($match),
    $pos);
1000.                } else if ( preg_match('/ {2,}/',$match) ) {
1001.                    $this->addCall('table_align', array($match),
    $pos);
1002.                } else if ( $match == "\n|" ) {
1003.                    $this->addCall('table_row', array(), $pos);
1004.                    $this->addCall('tablecell', array(), $pos);
1005.                } else if ( $match == "\n^" ) {
1006.                    $this->addCall('table_row', array(), $pos);
1007.                    $this->addCall('tableheader', array(), $pos);
1008.                } else if ( $match == '|' ) {
1009.                    $this->addCall('tablecell', array(), $pos);
1010.                } else if ( $match == '^' ) {
1011.                    $this->addCall('tableheader', array(), $pos);
1012.                }
1013.            break;
1014.        }
1015.        return true;
1016.    }
1017.
1018.    // endregion modes
1019. }
1020.
1021. //------------------------------------------------------------------
    --------
1022. function Doku_Handler_Parse_Media($match) {
1023.
1024.    // Strip the opening and closing markup
1025.    $link = preg_replace(array('/^\{\{/','/\}\}$/u'),'',$match);
1026.
1027.    // Split title from URL
1028.    $link = sexplode('|', $link, 2);
1029.
1030.    // Check alignment
1031.    $ralign = (bool)preg_match('/^ /',$link[0]);
1032.    $lalign = (bool)preg_match('/ $/',$link[0]);
1033.
1034.    // Logic = what's that ;)...
1035.    if ( $lalign & $ralign ) {
1036.        $align = 'center';
1037.    } else if ( $ralign ) {
1038.        $align = 'right';
1039.    } else if ( $lalign ) {
1040.        $align = 'left';
1041.    } else {
1042.        $align = null;
1043.    }
```

```
1044.
1045.     // The title...
1046.     if ( !isset($link[1]) ) {
1047.         $link[1] = null;
1048.     }
1049.
1050.     //remove aligning spaces
1051.     $link[0] = trim($link[0]);
1052.
1053.     //split into src and parameters (using the very last
     questionmark)
1054.     $pos = strrpos($link[0], '?');
1055.     if($pos !== false){
1056.         $src   = substr($link[0],0,$pos);
1057.         $param = substr($link[0],$pos+1);
1058.     }else{
1059.         $src   = $link[0];
1060.         $param = '';
1061.     }
1062.
1063.     //parse width and height
1064.     if(preg_match('#(\d+)(x(\d+))?#i',$param,$size)){
1065.         !empty($size[1]) ? $w = $size[1] : $w = null;
1066.         !empty($size[3]) ? $h = $size[3] : $h = null;
1067.     } else {
1068.         $w = null;
1069.         $h = null;
1070.     }
1071.
1072.     //get linking command
1073.     if(preg_match('/nolink/i',$param)){
1074.         $linking = 'nolink';
1075.     }else if(preg_match('/direct/i',$param)){
1076.         $linking = 'direct';
1077.     }else if(preg_match('/linkonly/i',$param)){
1078.         $linking = 'linkonly';
1079.     }else{
1080.         $linking = 'details';
1081.     }
1082.
1083.     //get caching command
1084.     if (preg_match('/(nocache|recache)/i',$param,$cachemode)){
1085.         $cache = $cachemode[1];
1086.     }else{
1087.         $cache = 'cache';
1088.     }
1089.
1090.     // Check whether this is a local or remote image or interwiki
1091.     if (media_isexternal($src) || link_isinterwiki($src)){
1092.         $call = 'externalmedia';
1093.     } else {
```

```
1094.             $call = 'internalmedia';
1095.         }
1096.
1097.         $params = array(
1098.             'type'=>$call,
1099.             'src'=>$src,
1100.             'title'=>$link[1],
1101.             'align'=>$align,
1102.             'width'=>$w,
1103.             'height'=>$h,
1104.             'cache'=>$cache,
1105.             'linking'=>$linking,
1106.         );
1107.
1108.         return $params;
1109.     }
```