```php
<?php

/**
 * DokuWiki template functions
 *
 * @license    GPL 2 (http://www.gnu.org/licenses/gpl.html)
 * @author     Andreas Gohr <andi@splitbrain.org>
 */

use dokuwiki\ActionRouter;
use dokuwiki\Action\Exception\FatalException;
use dokuwiki\Extension\PluginInterface;
use dokuwiki\Ui\Admin;
use dokuwiki\StyleUtils;
use dokuwiki\Menu\Item\AbstractItem;
use dokuwiki\Form\Form;
use dokuwiki\Menu\MobileMenu;
use dokuwiki\Ui\Subscribe;
use dokuwiki\Extension\AdminPlugin;
use dokuwiki\Extension\Event;
use dokuwiki\File\PageResolver;

/**
 * Access a template file
 *
 * Returns the path to the given file inside the current template, uses
 * default template if the custom version doesn't exist.
 *
 * @param string $file
 * @return string
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
function template($file)
{
    global $conf;

    if (@is_readable(DOKU_INC . 'lib/tpl/' . $conf['template'] . '/' . $file))
        return DOKU_INC . 'lib/tpl/' . $conf['template'] . '/' . $file;

    return DOKU_INC . 'lib/tpl/dokuwiki/' . $file;
}

/**
 * Convenience function to access template dir from local FS
 *
 * This replaces the deprecated DOKU_TPLINC constant
 *
 * @param string $tpl The template to use, default to current one
 * @return string
```

```
51  *
52  * @author Andreas Gohr <andi@splitbrain.org>
53  */
54 function tpl_incdir($tpl = '')
55 {
56     global $conf;
57     if (!$tpl) $tpl = $conf['template'];
58     return DOKU_INC . 'lib/tpl/' . $tpl . '/';
59 }
60
61 /**
62  * Convenience function to access template dir from web
63  *
64  * This replaces the deprecated DOKU_TPL constant
65  *
66  * @param string $tpl The template to use, default to current one
67  * @return string
68  *
69  * @author Andreas Gohr <andi@splitbrain.org>
70  */
71 function tpl_basedir($tpl = '')
72 {
73     global $conf;
74     if (!$tpl) $tpl = $conf['template'];
75     return DOKU_BASE . 'lib/tpl/' . $tpl . '/';
76 }
77
78 /**
79  * Print the content
80  *
81  * This function is used for printing all the usual content
82  * (defined by the global $ACT var) by calling the appropriate
83  * outputfunction(s) from html.php
84  *
85  * Everything that doesn't use the main template file isn't
86  * handled by this function. ACL stuff is not done here either.
87  *
88  * @param bool $prependTOC should the TOC be displayed here?
89  * @return bool true if any output
90  *
91  * @triggers TPL_ACT_RENDER
92  * @triggers TPL_CONTENT_DISPLAY
93  * @author Andreas Gohr <andi@splitbrain.org>
94  */
95 function tpl_content($prependTOC = true)
96 {
97     global $ACT;
98     global $INFO;
99     $INFO['prependTOC'] = $prependTOC;
100
```

```
101     ob_start();
102     Event::createAndTrigger('TPL_ACT_RENDER', $ACT, 'tpl_content_core');
103     $html_output = ob_get_clean();
104     Event::createAndTrigger('TPL_CONTENT_DISPLAY', $html_output,
function ($html_output) {
105         echo $html_output;
106     });
107
108     return !empty($html_output);
109 }
110
111 /**
112  * Default Action of TPL_ACT_RENDER
113  *
114  * @return bool
115  */
116 function tpl_content_core()
117 {
118     $router = ActionRouter::getInstance();
119     try {
120         $router->getAction()->tplContent();
121     } catch (FatalException $e) {
122         // there was no content for the action
123         msg(hsc($e->getMessage()), -1);
124         return false;
125     }
126     return true;
127 }
128
129 /**
130  * Places the TOC where the function is called
131  *
132  * If you use this you most probably want to call tpl_content with
133  * a false argument
134  *
135  * @param bool $return Should the TOC be returned instead to be printed?
136  * @return string
137  *
138  * @author Andreas Gohr <andi@splitbrain.org>
139  */
140 function tpl_toc($return = false)
141 {
142     global $TOC;
143     global $ACT;
144     global $ID;
145     global $REV;
146     global $INFO;
147     global $conf;
148     $toc = [];
149
150     if (is_array($TOC)) {
```

```
151         // if a TOC was prepared in global scope, always use it
152         $toc = $TOC;
153     } elseif (($ACT == 'show' || str_starts_with($ACT, 'export')) &&
!$REV && $INFO['exists']) {
154         // get TOC from metadata, render if neccessary
155         $meta = p_get_metadata($ID, '', METADATA_RENDER_USING_CACHE);
156         $tocok = $meta['internal']['toc'] ?? true;
157         $toc = $meta['description']['tableofcontents'] ?? null;
158         if (!$tocok || !is_array($toc) || !$conf['tocminheads'] ||
count($toc) < $conf['tocminheads']) {
159             $toc = [];
160         }
161     } elseif ($ACT == 'admin') {
162         // try to load admin plugin TOC
163         /** @var AdminPlugin $plugin */
164         if ($plugin = plugin_getRequestAdminPlugin()) {
165             $toc = $plugin->getTOC();
166             $TOC = $toc; // avoid later rebuild
167         }
168     }
169
170     Event::createAndTrigger('TPL_TOC_RENDER', $toc, null, false);
171     $html = html_TOC($toc);
172     if ($return) return $html;
173     echo $html;
174     return '';
175 }
176
177 /**
178  * Handle the admin page contents
179  *
180  * @return bool
181  *
182  * @author Andreas Gohr <andi@splitbrain.org>
183  */
184 function tpl_admin()
185 {
186     global $INFO;
187     global $TOC;
188     global $INPUT;
189
190     $plugin = null;
191     $class = $INPUT->str('page');
192     if (!empty($class)) {
193         $pluginlist = plugin_list('admin');
194
195         if (in_array($class, $pluginlist)) {
196             // attempt to load the plugin
197             /** @var AdminPlugin $plugin */
198             $plugin = plugin_load('admin', $class);
```

```php
        }
    }

    if ($plugin instanceof PluginInterface) {
        if (!is_array($TOC)) $TOC = $plugin->getTOC(); //if TOC wasn't requested yet
        if ($INFO['prependTOC']) tpl_toc();
        $plugin->html();
    } else {
        $admin = new Admin();
        $admin->show();
    }
    return true;
}

/**
 * Print the correct HTML meta headers
 *
 * This has to go into the head section of your template.
 *
 * @param bool $alt Should feeds and alternative format links be added?
 * @return bool
 * @throws JsonException
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 * @triggers TPL_METAHEADER_OUTPUT
 */
function tpl_metaheaders($alt = true)
{
    global $ID;
    global $REV;
    global $INFO;
    global $JSINFO;
    global $ACT;
    global $QUERY;
    global $lang;
    global $conf;
    global $updateVersion;
    /** @var Input $INPUT */
    global $INPUT;

    // prepare the head array
    $head = [];

    // prepare seed for js and css
    $tseed = $updateVersion;
    $depends = getConfigFiles('main');
    $depends[] = DOKU_CONF . "tpl/" . $conf['template'] . "/style.ini";
    foreach ($depends as $f) $tseed .= @filemtime($f);
    $tseed = md5($tseed);

```

```
249     // the usual stuff
250     $head['meta'][] = ['name' => 'generator', 'content' => 'DokuWiki'];
251     if (actionOK('search')) {
252         $head['link'][] = [
253             'rel' => 'search',
254             'type' => 'application/opensearchdescription+xml',
255             'href' => DOKU_BASE . 'lib/exe/opensearch.php',
256             'title' => $conf['title']
257         ];
258     }
259
260     $head['link'][] = ['rel' => 'start', 'href' => DOKU_BASE];
261     if (actionOK('index')) {
262         $head['link'][] = [
263             'rel' => 'contents',
264             'href' => wl($ID, 'do=index', false, '&'),
265             'title' => $lang['btn_index']
266         ];
267     }
268
269     if (actionOK('manifest')) {
270         $head['link'][] = [
271             'rel' => 'manifest',
272             'href' => DOKU_BASE . 'lib/exe/manifest.php'
273         ];
274     }
275
276     $styleUtil = new StyleUtils();
277     $styleIni = $styleUtil->cssStyleini();
278     $replacements = $styleIni['replacements'];
279     if (!empty($replacements['__theme_color__'])) {
280         $head['meta'][] = [
281             'name' => 'theme-color',
282             'content' => $replacements['__theme_color__']
283         ];
284     }
285
286     if ($alt) {
287         if (actionOK('rss')) {
288             $head['link'][] = [
289                 'rel' => 'alternate',
290                 'type' => 'application/rss+xml',
291                 'title' => $lang['btn_recent'],
292                 'href' => DOKU_BASE . 'feed.php'
293             ];
294             $head['link'][] = [
295                 'rel' => 'alternate',
296                 'type' => 'application/rss+xml',
297                 'title' => $lang['currentns'],
298                 'href' => DOKU_BASE . 'feed.php?mode=list&ns=' .
```

```
(isset($INFO) ? $INFO['namespace'] : '')
299             ];
300         }
301         if (($ACT == 'show' || $ACT == 'search') && $INFO['writable']) {
302             $head['link'][] = [
303                 'rel' => 'edit',
304                 'title' => $lang['btn_edit'],
305                 'href' => wl($ID, 'do=edit', false, '&')
306             ];
307         }
308 
309         if (actionOK('rss') && $ACT == 'search') {
310             $head['link'][] = [
311                 'rel' => 'alternate',
312                 'type' => 'application/rss+xml',
313                 'title' => $lang['searchresult'],
314                 'href' => DOKU_BASE . 'feed.php?mode=search&q=' . $QUERY
315             ];
316         }
317 
318         if (actionOK('export_xhtml')) {
319             $head['link'][] = [
320                 'rel' => 'alternate',
321                 'type' => 'text/html',
322                 'title' => $lang['plainhtml'],
323                 'href' => exportlink($ID, 'xhtml', '', false, '&')
324             ];
325         }
326 
327         if (actionOK('export_raw')) {
328             $head['link'][] = [
329                 'rel' => 'alternate',
330                 'type' => 'text/plain',
331                 'title' => $lang['wikimarkup'],
332                 'href' => exportlink($ID, 'raw', '', false, '&')
333             ];
334         }
335     }
336 
337     // setup robot tags appropriate for different modes
338     if (($ACT == 'show' || $ACT == 'export_xhtml') && !$REV) {
339         if ($INFO['exists']) {
340             //delay indexing:
341             if ((time() - $INFO['lastmod']) >= $conf['indexdelay'] &&
!isHiddenPage($ID)) {
342                 $head['meta'][] = ['name' => 'robots', 'content' =>
'index,follow'];
343             } else {
344                 $head['meta'][] = ['name' => 'robots', 'content' =>
'noindex,nofollow'];
345             }
```

```
346             $canonicalUrl = wl($ID, '', true, '&');
347             if ($ID == $conf['start']) {
348                 $canonicalUrl = DOKU_URL;
349             }
350             $head['link'][] = ['rel' => 'canonical', 'href' =>
$canonicalUrl];
351         } else {
352             $head['meta'][] = ['name' => 'robots', 'content' =>
'noindex,follow'];
353         }
354     } elseif (defined('DOKU_MEDIADETAIL')) {
355         $head['meta'][] = ['name' => 'robots', 'content' =>
'index,follow'];
356     } else {
357         $head['meta'][] = ['name' => 'robots', 'content' =>
'noindex,nofollow'];
358     }
359
360     // set metadata
361     if ($ACT == 'show' || $ACT == 'export_xhtml') {
362         // keywords (explicit or implicit)
363         if (!empty($INFO['meta']['subject'])) {
364             $head['meta'][] = ['name' => 'keywords', 'content' =>
implode(',', $INFO['meta']['subject'])];
365         } else {
366             $head['meta'][] = ['name' => 'keywords', 'content' =>
str_replace(':', ',', $ID)];
367         }
368     }
369
370     // load stylesheets
371     $head['link'][] = [
372         'rel' => 'stylesheet',
373         'href' => DOKU_BASE . 'lib/exe/css.php?t=' .
rawurlencode($conf['template']) . '&tseed=' . $tseed
374     ];
375
376     $script = "var NS='" . (isset($INFO) ? $INFO['namespace'] : '') .
"';";
377     if ($conf['useacl'] && $INPUT->server->str('REMOTE_USER')) {
378         $script .= "var SIG=" . toolbar_signature() . ";";
379     }
380     jsinfo();
381     $script .= 'var JSINFO = ' . json_encode($JSINFO,
JSON_THROW_ON_ERROR) . ';';
382     $script .= '(function(H){H.className=H.className.replace(/\bno-
js\b/,\'js\')})(document.documentElement);';
383     $head['script'][] = ['_data' => $script];
384
385     // load jquery
```

```
386     $jquery = getCdnUrls();
387     foreach ($jquery as $src) {
388         $head['script'][] = [
389                 '_data' => '',
390                 'src' => $src
391             ] + ($conf['defer_js'] ? ['defer' => 'defer'] : []);
392     }
393
394     // load our javascript dispatcher
395     $head['script'][] = [
396             '_data' => '',
397             'src' => DOKU_BASE . 'lib/exe/js.php' . '?t=' .
rawurlencode($conf['template']) . '&tseed=' . $tseed
398         ] + ($conf['defer_js'] ? ['defer' => 'defer'] : []);
399
400     // trigger event here
401     Event::createAndTrigger('TPL_METAHEADER_OUTPUT', $head,
'_tpl_metaheaders_action', true);
402     return true;
403 }
404
405 /**
406  * prints the array build by tpl_metaheaders
407  *
408  * $data is an array of different header tags. Each tag can have
multiple
409  * instances. Attributes are given as key value pairs. Values will be
HTML
410  * encoded automatically so they should be provided as is in the $data
array.
411  *
412  * For tags having a body attribute specify the body data in the special
413  * attribute '_data'. This field will NOT BE ESCAPED automatically.
414  *
415  * Inline scripts will use any nonce provided in the environment
variable 'NONCE'.
416  *
417  * @param array $data
418  *
419  * @author Andreas Gohr <andi@splitbrain.org>
420  */
421 function _tpl_metaheaders_action($data)
422 {
423     $nonce = getenv('NONCE');
424     foreach ($data as $tag => $inst) {
425         foreach ($inst as $attr) {
426             if (empty($attr)) {
427                 continue;
428             }
429             if ($nonce && $tag == 'script' && !empty($attr['_data'])) {
430                 $attr['nonce'] = $nonce; // add nonce to inline script
```

```
tags
431             }
432             echo '<', $tag, ' ', buildAttributes($attr);
433             if (isset($attr['_data']) || $tag == 'script') {
434                 echo '>', $attr['_data'] ?? '', '</', $tag, '>';
435             } else {
436                 echo '/>';
437             }
438             echo "\n";
439         }
440     }
441 }
442
443 /**
444  * Output the given script as inline script tag
445  *
446  * This function will add the nonce attribute if a nonce is available.
447  *
448  * The script is NOT automatically escaped!
449  *
450  * @param string $script
451  * @param bool $return Return or print directly?
452  * @return string|void
453  */
454 function tpl_inlineScript($script, $return = false)
455 {
456     $nonce = getenv('NONCE');
457     if ($nonce) {
458         $script = '<script nonce="' . $nonce . '">' . $script .
'</script>';
459     } else {
460         $script = '<script>' . $script . '</script>';
461     }
462
463     if ($return) return $script;
464     echo $script;
465 }
466
467 /**
468  * Print a link
469  *
470  * Just builds a link.
471  *
472  * @param string $url
473  * @param string $name
474  * @param string $more
475  * @param bool $return if true return the link html, otherwise print
476  * @return bool|string html of the link, or true if printed
477  *
478  * @author Andreas Gohr <andi@splitbrain.org>
```

```php
 */
function tpl_link($url, $name, $more = '', $return = false)
{
    $out = '<a href="' . $url . '" ';
    if ($more) $out .= ' ' . $more;
    $out .= ">$name</a>";
    if ($return) return $out;
    echo $out;
    return true;
}

/**
 * Prints a link to a WikiPage
 *
 * Wrapper around html_wikilink
 *
 * @param string $id page id
 * @param string|null $name the name of the link
 * @param bool $return
 * @return true|string
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
function tpl_pagelink($id, $name = null, $return = false)
{
    $out = '<bdi>' . html_wikilink($id, $name) . '</bdi>';
    if ($return) return $out;
    echo $out;
    return true;
}

/**
 * get the parent page
 *
 * Tries to find out which page is parent.
 * returns false if none is available
 *
 * @param string $id page id
 * @return false|string
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
function tpl_getparent($id)
{
    $resolver = new PageResolver('root');

    $parent = getNS($id) . ':';
    $parent = $resolver->resolveId($parent);
    if ($parent == $id) {
        $pos = strrpos(getNS($id), ':');
        $parent = substr($parent, 0, $pos) . ':';
```

```
530         $parent = $resolver->resolveId($parent);
531         if ($parent == $id) return false;
532     }
533     return $parent;
534 }
535
536 /**
537  * Print one of the buttons
538  *
539  * @param string $type
540  * @param bool $return
541  * @return bool|string html, or false if no data, true if printed
542  * @see    tpl_get_action
543  *
544  * @author Adrian Lang <mail@adrianlang.de>
545  * @deprecated 2017-09-01 see devel:menus
546  */
547 function tpl_button($type, $return = false)
548 {
549     dbg_deprecated('see devel:menus');
550     $data = tpl_get_action($type);
551     if ($data === false) {
552         return false;
553     } elseif (!is_array($data)) {
554         $out = sprintf($data, 'button');
555     } else {
556         /**
557          * @var string $accesskey
558          * @var string $id
559          * @var string $method
560          * @var array $params
561          */
562         extract($data);
563         if ($id === '#dokuwiki__top') {
564             $out = html_topbtn();
565         } else {
566             $out = html_btn($type, $id, $accesskey, $params, $method);
567         }
568     }
569     if ($return) return $out;
570     echo $out;
571     return true;
572 }
573
574 /**
575  * Like the action buttons but links
576  *
577  * @param string $type action command
578  * @param string $pre prefix of link
579  * @param string $suf suffix of link
```

```php
 * @param string $inner innerHML of link
 * @param bool $return if true it returns html, otherwise prints
 * @return bool|string html or false if no data, true if printed
 *
 * @see    tpl_get_action
 * @author Adrian Lang <mail@adrianlang.de>
 * @deprecated 2017-09-01 see devel:menus
 */
function tpl_actionlink($type, $pre = '', $suf = '', $inner = '',
$return = false)
{
    dbg_deprecated('see devel:menus');
    global $lang;
    $data = tpl_get_action($type);
    if ($data === false) {
        return false;
    } elseif (!is_array($data)) {
        $out = sprintf($data, 'link');
    } else {
        /**
         * @var string $accesskey
         * @var string $id
         * @var string $method
         * @var bool $nofollow
         * @var array $params
         * @var string $replacement
         */
        extract($data);
        if (strpos($id, '#') === 0) {
            $linktarget = $id;
        } else {
            $linktarget = wl($id, $params);
        }
        $caption = $lang['btn_' . $type];
        if (strpos($caption, '%s')) {
            $caption = sprintf($caption, $replacement);
        }
        $akey = '';
        $addTitle = '';
        if ($accesskey) {
            $akey = 'accesskey="' . $accesskey . '" ';
            $addTitle = ' [' . strtoupper($accesskey) . ']';
        }
        $rel = $nofollow ? 'rel="nofollow" ' : '';
        $out = tpl_link(
            $linktarget,
            $pre . ($inner ?: $caption) . $suf,
            'class="action ' . $type . '" ' .
            $akey . $rel .
            'title="' . hsc($caption) . $addTitle . '"',
            true
```

```
630         );
631     }
632     if ($return) return $out;
633     echo $out;
634     return true;
635 }
636 
637 /**
638  * Check the actions and get data for buttons and links
639  *
640  * @param string $type
641  * @return array|bool|string
642  *
643  * @author Adrian Lang <mail@adrianlang.de>
644  * @author Andreas Gohr <andi@splitbrain.org>
645  * @author Matthias Grimm <matthiasgrimm@users.sourceforge.net>
646  * @deprecated 2017-09-01 see devel:menus
647  */
648 function tpl_get_action($type)
649 {
650     dbg_deprecated('see devel:menus');
651     if ($type == 'history') $type = 'revisions';
652     if ($type == 'subscription') $type = 'subscribe';
653     if ($type == 'img_backto') $type = 'imgBackto';
654 
655     $class = '\\dokuwiki\\Menu\\Item\\' . ucfirst($type);
656     if (class_exists($class)) {
657         try {
658             /** @var AbstractItem $item */
659             $item = new $class();
660             $data = $item->getLegacyData();
661             $unknown = false;
662         } catch (RuntimeException $ignored) {
663             return false;
664         }
665     } else {
666         global $ID;
667         $data = [
668             'accesskey' => null,
669             'type' => $type,
670             'id' => $ID,
671             'method' => 'get',
672             'params' => ['do' => $type],
673             'nofollow' => true,
674             'replacement' => ''
675         ];
676         $unknown = true;
677     }
678 
679     $evt = new Event('TPL_ACTION_GET', $data);
```

```
680     if ($evt->advise_before()) {
681         //handle unknown types
682         if ($unknown) {
683             $data = '[unknown %s type]';
684         }
685     }
686     $evt->advise_after();
687     unset($evt);
688 
689     return $data;
690 }
691 
692 /**
693  * Wrapper around tpl_button() and tpl_actionlink()
694  *
695  * @param string $type action command
696  * @param bool $link link or form button?
697  * @param string|bool $wrapper HTML element wrapper
698  * @param bool $return return or print
699  * @param string $pre prefix for links
700  * @param string $suf suffix for links
701  * @param string $inner inner HTML for links
702  * @return bool|string
703  *
704  * @author Anika Henke <anika@selfthinker.org>
705  * @deprecated 2017-09-01 see devel:menus
706  */
707 function tpl_action($type, $link = false, $wrapper = false, $return =
false, $pre = '', $suf = '', $inner = '')
708 {
709     dbg_deprecated('see devel:menus');
710     $out = '';
711     if ($link) {
712         $out .= tpl_actionlink($type, $pre, $suf, $inner, true);
713     } else {
714         $out .= tpl_button($type, true);
715     }
716     if ($out && $wrapper) $out = "<$wrapper>$out</$wrapper>";
717 
718     if ($return) return $out;
719     echo $out;
720     return (bool)$out;
721 }
722 
723 /**
724  * Print the search form
725  *
726  * If the first parameter is given a div with the ID 'qsearch_out' will
727  * be added which instructs the ajax pagequicksearch to kick in and
place
728  * its output into this div. The second parameter controls the
```

```
proprietary
729  * attribute autocomplete. If set to false this attribute will be set
with an
730  * value of "off" to instruct the browser to disable it's own built in
731  * autocompletion feature (MSIE and Firefox)
732  *
733  * @param bool $ajax
734  * @param bool $autocomplete
735  * @return bool
736  *
737  * @author Andreas Gohr <andi@splitbrain.org>
738  */
739 function tpl_searchform($ajax = true, $autocomplete = true)
740 {
741     global $lang;
742     global $ACT;
743     global $QUERY;
744     global $ID;
745
746     // don't print the search form if search action has been disabled
747     if (!actionOK('search')) return false;
748
749     $searchForm = new Form([
750         'action' => wl(),
751         'method' => 'get',
752         'role' => 'search',
753         'class' => 'search',
754         'id' => 'dw__search',
755     ], true);
756     $searchForm->addTagOpen('div')->addClass('no');
757     $searchForm->setHiddenField('do', 'search');
758     $searchForm->setHiddenField('id', $ID);
759     $searchForm->addTextInput('q')
760         ->addClass('edit')
761         ->attrs([
762             'title' => '[F]',
763             'accesskey' => 'f',
764             'placeholder' => $lang['btn_search'],
765             'autocomplete' => $autocomplete ? 'on' : 'off',
766         ])
767         ->id('qsearch__in')
768         ->val($ACT === 'search' ? $QUERY : '')
769         ->useInput(false);
770     $searchForm->addButton('', $lang['btn_search'])->attrs([
771         'type' => 'submit',
772         'title' => $lang['btn_search'],
773     ]);
774     if ($ajax) {
775
$searchForm->addTagOpen('div')->id('qsearch__out')->addClass('ajax_qsearch
```

```
JSpopup');
776         $searchForm->addTagClose('div');
777     }
778     $searchForm->addTagClose('div');
779
780     echo $searchForm->toHTML('QuickSearch');
781
782     return true;
783 }
784
785 /**
786  * Print the breadcrumbs trace
787  *
788  * @param string $sep Separator between entries
789  * @param bool $return return or print
790  * @return bool|string
791  *
792  * @author Andreas Gohr <andi@splitbrain.org>
793  */
794 function tpl_breadcrumbs($sep = null, $return = false)
795 {
796     global $lang;
797     global $conf;
798
799     //check if enabled
800     if (!$conf['breadcrumbs']) return false;
801
802     //set default
803     if (is_null($sep)) $sep = '•';
804
805     $out = '';
806
807     $crumbs = breadcrumbs(); //setup crumb trace
808
809     $crumbs_sep = ' <span class="bcsep">' . $sep . '</span> ';
810
811     //render crumbs, highlight the last one
812     $out .= '<span class="bchead">' . $lang['breadcrumb'] . '</span>';
813     $last = count($crumbs);
814     $i = 0;
815     foreach ($crumbs as $id => $name) {
816         $i++;
817         $out .= $crumbs_sep;
818         if ($i == $last) $out .= '<span class="curid">';
819         $out .= '<bdi>' . tpl_link(wl($id), hsc($name),
'class="breadcrumbs" title="' . $id . '"', true) . '</bdi>';
820         if ($i == $last) $out .= '</span>';
821     }
822     if ($return) return $out;
823     echo $out;
824     return (bool)$out;
```

```php
}

/**
 * Hierarchical breadcrumbs
 *
 * This code was suggested as replacement for the usual breadcrumbs.
 * It only makes sense with a deep site structure.
 *
 * @param string $sep Separator between entries
 * @param bool $return return or print
 * @return bool|string
 *
 * @todo   May behave strangely in RTL languages
 * @author <fredrik@averpil.com>
 * @author Andreas Gohr <andi@splitbrain.org>
 * @author Nigel McNie <oracle.shinoda@gmail.com>
 * @author Sean Coates <sean@caedmon.net>
 */
function tpl_youarehere($sep = null, $return = false)
{
    global $conf;
    global $ID;
    global $lang;

    // check if enabled
    if (!$conf['youarehere']) return false;

    //set default
    if (is_null($sep)) $sep = ' » ';

    $out = '';

    $parts = explode(':', $ID);
    $count = count($parts);

    $out .= '<span class="bchead">' . $lang['youarehere'] . ' </span>';

    // always print the startpage
    $out .= '<span class="home">' . tpl_pagelink(':' . $conf['start'],
null, true) . '</span>';

    // print intermediate namespace links
    $part = '';
    for ($i = 0; $i < $count - 1; $i++) {
        $part .= $parts[$i] . ':';
        $page = $part;
        if ($page == $conf['start']) continue; // Skip startpage

        // output
        $out .= $sep . tpl_pagelink($page, null, true);
```

```
874     }
875
876     // print current page, skipping start page, skipping for namespace
index
877     if (isset($page)) {
878         $page = (new PageResolver('root'))->resolveId($page);
879         if ($page == $part . $parts[$i]) {
880             if ($return) return $out;
881             echo $out;
882             return true;
883         }
884     }
885     $page = $part . $parts[$i];
886     if ($page == $conf['start']) {
887         if ($return) return $out;
888         echo $out;
889         return true;
890     }
891     $out .= $sep;
892     $out .= tpl_pagelink($page, null, true);
893     if ($return) return $out;
894     echo $out;
895     return (bool)$out;
896 }
897
898 /**
899  * Print info if the user is logged in
900  * and show full name in that case
901  *
902  * Could be enhanced with a profile link in future?
903  *
904  * @return bool
905  *
906  * @author Andreas Gohr <andi@splitbrain.org>
907  */
908 function tpl_userinfo()
909 {
910     global $lang;
911     /** @var Input $INPUT */
912     global $INPUT;
913
914     if ($INPUT->server->str('REMOTE_USER')) {
915         echo $lang['loggedinas'] . ' ' . userlink();
916         return true;
917     }
918     return false;
919 }
920
921 /**
922  * Print some info about the current page
923  *
```

```php
 * @param bool $ret return content instead of printing it
 * @return bool|string
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
function tpl_pageinfo($ret = false)
{
    global $conf;
    global $lang;
    global $INFO;
    global $ID;

    // return if we are not allowed to view the page
    if (!auth_quickaclcheck($ID)) {
        return false;
    }

    // prepare date and path
    $fn = $INFO['filepath'];
    if (!$conf['fullpath']) {
        if ($INFO['rev']) {
            $fn = str_replace($conf['olddir'] . '/', '', $fn);
        } else {
            $fn = str_replace($conf['datadir'] . '/', '', $fn);
        }
    }
    $fn = utf8_decodeFN($fn);
    $date = dformat($INFO['lastmod']);

    // print it
    if ($INFO['exists']) {
        $out = '<bdi>' . $fn . '</bdi>';
        $out .= ' · ';
        $out .= $lang['lastmod'];
        $out .= ' ';
        $out .= $date;
        if ($INFO['editor']) {
            $out .= ' ' . $lang['by'] . ' ';
            $out .= '<bdi>' . editorinfo($INFO['editor']) . '</bdi>';
        } else {
            $out .= ' (' . $lang['external_edit'] . ')';
        }
        if ($INFO['locked']) {
            $out .= ' · ';
            $out .= $lang['lockedby'];
            $out .= ' ';
            $out .= '<bdi>' . editorinfo($INFO['locked']) . '</bdi>';
        }
        if ($ret) {
            return $out;
```

```
974         } else {
975             echo $out;
976             return true;
977         }
978     }
979     return false;
980 }
981
982 /**
983  * Prints or returns the name of the given page (current one if none
given).
984  *
985  * If useheading is enabled this will use the first headline else
986  * the given ID is used.
987  *
988  * @param string $id page id
989  * @param bool $ret return content instead of printing
990  * @return bool|string
991  *
992  * @author Andreas Gohr <andi@splitbrain.org>
993  */
994 function tpl_pagetitle($id = null, $ret = false)
995 {
996     global $ACT, $conf, $lang;
997
998     if (is_null($id)) {
999         global $ID;
1000         $id = $ID;
1001     }
1002
1003     $name = $id;
1004     if (useHeading('navigation')) {
1005         $first_heading = p_get_first_heading($id);
1006         if ($first_heading) $name = $first_heading;
1007     }
1008
1009     // default page title is the page name, modify with the current
action
1010     switch ($ACT) {
1011         // admin functions
1012         case 'admin':
1013             $page_title = $lang['btn_admin'];
1014             // try to get the plugin name
1015             /** @var AdminPlugin $plugin */
1016             if ($plugin = plugin_getRequestAdminPlugin()) {
1017                 $plugin_title = $plugin->getMenuText($conf['lang']);
1018                 $page_title = $plugin_title ?:
$plugin->getPluginName();
1019             }
1020             break;
1021
```

```
1022         // show action as title
1023         case 'login':
1024         case 'profile':
1025         case 'register':
1026         case 'resendpwd':
1027         case 'index':
1028         case 'search':
1029             $page_title = $lang['btn_' . $ACT];
1030             break;
1031 
1032         // add pen during editing
1033         case 'edit':
1034         case 'preview':
1035             $page_title = "✎ " . $name;
1036             break;
1037 
1038         // add action to page name
1039         case 'revisions':
1040             $page_title = $name . ' - ' . $lang['btn_revs'];
1041             break;
1042 
1043         // add action to page name
1044         case 'backlink':
1045         case 'recent':
1046         case 'subscribe':
1047             $page_title = $name . ' - ' . $lang['btn_' . $ACT];
1048             break;
1049 
1050         default: // SHOW and anything else not included
1051             $page_title = $name;
1052     }
1053 
1054     if ($ret) {
1055         return hsc($page_title);
1056     } else {
1057         echo hsc($page_title);
1058         return true;
1059     }
1060 }
1061 
1062 /**
1063  * Returns the requested EXIF/IPTC tag from the current image
1064  *
1065  * If $tags is an array all given tags are tried until a
1066  * value is found. If no value is found $alt is returned.
1067  *
1068  * Which texts are known is defined in the functions _exifTagNames
1069  * and _iptcTagNames() in inc/jpeg.php (You need to prepend IPTC
1070  * to the names of the latter one)
1071  *
```

```php
 * Only allowed in: detail.php
 *
 * @param array|string $tags tag or array of tags to try
 * @param string $alt alternative output if no data was found
 * @param null|string $src the image src, uses global $SRC if not given
 * @return string
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
function tpl_img_getTag($tags, $alt = '', $src = null)
{
    // Init Exif Reader
    global $SRC, $imgMeta;

    if (is_null($src)) $src = $SRC;
    if (is_null($src)) return $alt;

    if (!isset($imgMeta)) {
        $imgMeta = new JpegMeta($src);
    }
    if ($imgMeta === false) return $alt;
    $info = cleanText($imgMeta->getField($tags));
    if (!$info) return $alt;
    return $info;
}


/**
 * Garbage collects up the open JpegMeta object.
 */
function tpl_img_close()
{
    global $imgMeta;
    $imgMeta = null;
}

/**
 * Prints a html description list of the metatags of the current image
 */
function tpl_img_meta()
{
    global $lang;

    $tags = tpl_get_img_meta();

    echo '<dl>';
    foreach ($tags as $tag) {
        $label = $lang[$tag['langkey']];
        if (!$label) $label = $tag['langkey'] . ':';

        echo '<dt>' . $label . '</dt><dd>';
```

```
1123         if ($tag['type'] == 'date') {
1124             echo dformat($tag['value']);
1125         } else {
1126             echo hsc($tag['value']);
1127         }
1128         echo '</dd>';
1129     }
1130     echo '</dl>';
1131 }
1132
1133 /**
1134  * Returns metadata as configured in mediameta config file, ready for
creating html
1135  *
1136  * @return array with arrays containing the entries:
1137  *   - string langkey  key to lookup in the $lang var, if not found
printed as is
1138  *   - string type     type of value
1139  *   - string value    tag value (unescaped)
1140  */
1141 function tpl_get_img_meta()
1142 {
1143
1144     $config_files = getConfigFiles('mediameta');
1145     foreach ($config_files as $config_file) {
1146         if (file_exists($config_file)) {
1147             include($config_file);
1148         }
1149     }
1150     $tags = [];
1151     foreach ($fields as $tag) {
1152         $t = [];
1153         if (!empty($tag[0])) {
1154             $t = [$tag[0]];
1155         }
1156         if (isset($tag[3]) && is_array($tag[3])) {
1157             $t = array_merge($t, $tag[3]);
1158         }
1159         $value = tpl_img_getTag($t);
1160         if ($value) {
1161             $tags[] = ['langkey' => $tag[1], 'type' => $tag[2], 'value'
=> $value];
1162         }
1163     }
1164     return $tags;
1165 }
1166
1167 /**
1168  * Prints the image with a link to the full sized version
1169  *
```

```
1170  * Only allowed in: detail.php
1171  *
1172  * @triggers TPL_IMG_DISPLAY
1173  * @param int $maxwidth - maximal width of the image
1174  * @param int $maxheight - maximal height of the image
1175  * @param bool $link - link to the orginal size?
1176  * @param array $params - additional image attributes
1177  * @return bool Result of TPL_IMG_DISPLAY
1178  */
1179 function tpl_img($maxwidth = 0, $maxheight = 0, $link = true, $params =
null)
1180 {
1181     global $IMG;
1182     /** @var Input $INPUT */
1183     global $INPUT;
1184     global $REV;
1185     $w = (int)tpl_img_getTag('File.Width');
1186     $h = (int)tpl_img_getTag('File.Height');
1187
1188     //resize to given max values
1189     $ratio = 1;
1190     if ($w >= $h) {
1191         if ($maxwidth && $w >= $maxwidth) {
1192             $ratio = $maxwidth / $w;
1193         } elseif ($maxheight && $h > $maxheight) {
1194             $ratio = $maxheight / $h;
1195         }
1196     } elseif ($maxheight && $h >= $maxheight) {
1197         $ratio = $maxheight / $h;
1198     } elseif ($maxwidth && $w > $maxwidth) {
1199         $ratio = $maxwidth / $w;
1200     }
1201     if ($ratio) {
1202         $w = floor($ratio * $w);
1203         $h = floor($ratio * $h);
1204     }
1205
1206     //prepare URLs
1207     $url = ml($IMG, ['cache' => $INPUT->str('cache'), 'rev' => $REV],
true, '&');
1208     $src = ml($IMG, ['cache' => $INPUT->str('cache'), 'rev' => $REV,
'w' => $w, 'h' => $h], true, '&');
1209
1210     //prepare attributes
1211     $alt = tpl_img_getTag('Simple.Title');
1212     if (is_null($params)) {
1213         $p = [];
1214     } else {
1215         $p = $params;
1216     }
1217     if ($w) $p['width'] = $w;
```

```
1218     if ($h) $p['height'] = $h;
1219     $p['class'] = 'img_detail';
1220     if ($alt) {
1221         $p['alt'] = $alt;
1222         $p['title'] = $alt;
1223     } else {
1224         $p['alt'] = '';
1225     }
1226     $p['src'] = $src;
1227 
1228     $data = ['url' => ($link ? $url : null), 'params' => $p];
1229     return Event::createAndTrigger('TPL_IMG_DISPLAY', $data,
'_tpl_img_action', true);
1230 }
1231 
1232 /**
1233  * Default action for TPL_IMG_DISPLAY
1234  *
1235  * @param array $data
1236  * @return bool
1237  */
1238 function _tpl_img_action($data)
1239 {
1240     global $lang;
1241     $p = buildAttributes($data['params']);
1242 
1243     if ($data['url']) echo '<a href="' . hsc($data['url']) . '"
title="' . $lang['mediaview'] . '">';
1244     echo '<img ' . $p . '/>';
1245     if ($data['url']) echo '</a>';
1246     return true;
1247 }
1248 
1249 /**
1250  * This function inserts a small gif which in reality is the indexer
function.
1251  *
1252  * Should be called somewhere at the very end of the main.php template
1253  *
1254  * @return bool
1255  */
1256 function tpl_indexerWebBug()
1257 {
1258     global $ID;
1259 
1260     $p = [];
1261     $p['src'] = DOKU_BASE . 'lib/exe/taskrunner.php?id=' .
rawurlencode($ID) .
1262         '&' . time();
1263     $p['width'] = 2; //no more 1x1 px image because we live in times of
```

```
ad blockers...
1264     $p['height'] = 1;
1265     $p['alt'] = '';
1266     $att = buildAttributes($p);
1267     echo "<img $att />";
1268     return true;
1269 }
1270 
1271 /**
1272  * tpl_getConf($id)
1273  *
1274  * use this function to access template configuration variables
1275  *
1276  * @param string $id name of the value to access
1277  * @param mixed $notset what to return if the setting is not available
1278  * @return mixed
1279  */
1280 function tpl_getConf($id, $notset = false)
1281 {
1282     global $conf;
1283     static $tpl_configloaded = false;
1284 
1285     $tpl = $conf['template'];
1286 
1287     if (!$tpl_configloaded) {
1288         $tconf = tpl_loadConfig();
1289         if ($tconf !== false) {
1290             foreach ($tconf as $key => $value) {
1291                 if (isset($conf['tpl'][$tpl][$key])) continue;
1292                 $conf['tpl'][$tpl][$key] = $value;
1293             }
1294             $tpl_configloaded = true;
1295         }
1296     }
1297 
1298     return $conf['tpl'][$tpl][$id] ?? $notset;
1299 }
1300 
1301 /**
1302  * tpl_loadConfig()
1303  *
1304  * reads all template configuration variables
1305  * this function is automatically called by tpl_getConf()
1306  *
1307  * @return false|array
1308  */
1309 function tpl_loadConfig()
1310 {
1311 
1312     $file = tpl_incdir() . '/conf/default.php';
1313     $conf = [];
```

```
1314
1315     if (!file_exists($file)) return false;
1316
1317     // load default config file
1318     include($file);
1319
1320     return $conf;
1321 }
1322
1323 // language methods
1324
1325 /**
1326  * tpl_getLang($id)
1327  *
1328  * use this function to access template language variables
1329  *
1330  * @param string $id key of language string
1331  * @return string
1332  */
1333 function tpl_getLang($id)
1334 {
1335     static $lang = [];
1336
1337     if (count($lang) === 0) {
1338         global $conf, $config_cascade; // definitely don't invoke
"global $lang"
1339
1340         $path = tpl_incdir() . 'lang/';
1341
1342         $lang = [];
1343
1344         // don't include once
1345         @include($path . 'en/lang.php');
1346         foreach ($config_cascade['lang']['template'] as $config_file) {
1347             if (file_exists($config_file . $conf['template'] .
'/en/lang.php')) {
1348                 include($config_file . $conf['template'] .
'/en/lang.php');
1349             }
1350         }
1351
1352         if ($conf['lang'] != 'en') {
1353             @include($path . $conf['lang'] . '/lang.php');
1354             foreach ($config_cascade['lang']['template'] as
$config_file) {
1355                 if (file_exists($config_file . $conf['template'] . '/'
. $conf['lang'] . '/lang.php')) {
1356                     include($config_file . $conf['template'] . '/' .
$conf['lang'] . '/lang.php');
1357                 }
```

```
1358             }
1359         }
1360     }
1361     return $lang[$id] ?? '';
1362 }
1363
1364 /**
1365  * Retrieve a language dependent file and pass to xhtml renderer for
display
1366  * template equivalent of p_locale_xhtml()
1367  *
1368  * @param string $id id of language dependent wiki page
1369  * @return  string     parsed contents of the wiki page in xhtml format
1370  */
1371 function tpl_locale_xhtml($id)
1372 {
1373     return p_cached_output(tpl_localeFN($id));
1374 }
1375
1376 /**
1377  * Prepends appropriate path for a language dependent filename
1378  *
1379  * @param string $id id of localized text
1380  * @return string wiki text
1381  */
1382 function tpl_localeFN($id)
1383 {
1384     $path = tpl_incdir() . 'lang/';
1385     global $conf;
1386     $file = DOKU_CONF . 'template_lang/' . $conf['template'] . '/' .
$conf['lang'] . '/' . $id . '.txt';
1387     if (!file_exists($file)) {
1388         $file = $path . $conf['lang'] . '/' . $id . '.txt';
1389         if (!file_exists($file)) {
1390             //fall back to english
1391             $file = $path . 'en/' . $id . '.txt';
1392         }
1393     }
1394     return $file;
1395 }
1396
1397 /**
1398  * prints the "main content" in the mediamanager popup
1399  *
1400  * Depending on the user's actions this may be a list of
1401  * files in a namespace, the meta editing dialog or
1402  * a message of referencing pages
1403  *
1404  * Only allowed in mediamanager.php
1405  *
1406  * @triggers MEDIAMANAGER_CONTENT_OUTPUT
```

```
1407  * @param bool $fromajax - set true when calling this function via ajax
1408  * @param string $sort
1409  *
1410  * @author Andreas Gohr <andi@splitbrain.org>
1411  */
1412 function tpl_mediaContent($fromajax = false, $sort = 'natural')
1413 {
1414     global $IMG;
1415     global $AUTH;
1416     global $INUSE;
1417     global $NS;
1418     global $JUMPTO;
1419     /** @var Input $INPUT */
1420     global $INPUT;
1421
1422     $do = $INPUT->extract('do')->str('do');
1423     if (in_array($do, ['save', 'cancel'])) $do = '';
1424
1425     if (!$do) {
1426         if ($INPUT->bool('edit')) {
1427             $do = 'metaform';
1428         } elseif (is_array($INUSE)) {
1429             $do = 'filesinuse';
1430         } else {
1431             $do = 'filelist';
1432         }
1433     }
1434
1435     // output the content pane, wrapped in an event.
1436     if (!$fromajax) echo '<div id="media__content">';
1437     $data = ['do' => $do];
1438     $evt = new Event('MEDIAMANAGER_CONTENT_OUTPUT', $data);
1439     if ($evt->advise_before()) {
1440         $do = $data['do'];
1441         if ($do == 'filesinuse') {
1442             media_filesinuse($INUSE, $IMG);
1443         } elseif ($do == 'filelist') {
1444             media_filelist($NS, $AUTH, $JUMPTO, false, $sort);
1445         } elseif ($do == 'searchlist') {
1446             media_searchlist($INPUT->str('q'), $NS, $AUTH);
1447         } else {
1448             msg('Unknown action ' . hsc($do), -1);
1449         }
1450     }
1451     $evt->advise_after();
1452     unset($evt);
1453     if (!$fromajax) echo '</div>';
1454 }
1455
1456 /**
```

```php
1457  * Prints the central column in full-screen media manager
1458  * Depending on the opened tab this may be a list of
1459  * files in a namespace, upload form or search form
1460  *
1461  * @author Kate Arzamastseva <pshns@ukr.net>
1462  */
1463 function tpl_mediaFileList()
1464 {
1465     global $AUTH;
1466     global $NS;
1467     global $JUMPTO;
1468     global $lang;
1469     /** @var Input $INPUT */
1470     global $INPUT;
1471
1472     $opened_tab = $INPUT->str('tab_files');
1473     if (!$opened_tab || !in_array($opened_tab, ['files', 'upload',
'search'])) $opened_tab = 'files';
1474     if ($INPUT->str('mediado') == 'update') $opened_tab = 'upload';
1475
1476     echo '<h2 class="a11y">' . $lang['mediaselect'] . '</h2>' . NL;
1477
1478     media_tabs_files($opened_tab);
1479
1480     echo '<div class="panelHeader">' . NL;
1481     echo '<h3>';
1482     $tabTitle = $NS ?: '[' . $lang['mediaroot'] . ']';
1483     printf($lang['media_' . $opened_tab], '<strong>' . hsc($tabTitle) .
'</strong>');
1484     echo '</h3>' . NL;
1485     if ($opened_tab === 'search' || $opened_tab === 'files') {
1486         media_tab_files_options();
1487     }
1488     echo '</div>' . NL;
1489
1490     echo '<div class="panelContent">' . NL;
1491     if ($opened_tab == 'files') {
1492         media_tab_files($NS, $AUTH, $JUMPTO);
1493     } elseif ($opened_tab == 'upload') {
1494         media_tab_upload($NS, $AUTH, $JUMPTO);
1495     } elseif ($opened_tab == 'search') {
1496         media_tab_search($NS, $AUTH);
1497     }
1498     echo '</div>' . NL;
1499 }
1500
1501 /**
1502  * Prints the third column in full-screen media manager
1503  * Depending on the opened tab this may be details of the
1504  * selected file, the meta editing dialog or
1505  * list of file revisions
```

```
1506  *
1507  * @param string $image
1508  * @param boolean $rev
1509  *
1510  * @author Kate Arzamastseva <pshns@ukr.net>
1511  */
1512 function tpl_mediaFileDetails($image, $rev)
1513 {
1514     global $conf, $DEL, $lang;
1515     /** @var Input $INPUT */
1516     global $INPUT;
1517
1518     $removed = (
1519         !file_exists(mediaFN($image)) &&
1520         file_exists(mediaMetaFN($image, '.changes')) &&
1521         $conf['mediarevisions']
1522     );
1523     if (!$image || (!file_exists(mediaFN($image)) && !$removed) ||
$DEL) return;
1524     if ($rev && !file_exists(mediaFN($image, $rev))) $rev = false;
1525     $ns = getNS($image);
1526     $do = $INPUT->str('mediado');
1527
1528     $opened_tab = $INPUT->str('tab_details');
1529
1530     $tab_array = ['view'];
1531     [, $mime] = mimetype($image);
1532     if ($mime == 'image/jpeg') {
1533         $tab_array[] = 'edit';
1534     }
1535     if ($conf['mediarevisions']) {
1536         $tab_array[] = 'history';
1537     }
1538
1539     if (!$opened_tab || !in_array($opened_tab, $tab_array)) $opened_tab
= 'view';
1540     if ($INPUT->bool('edit')) $opened_tab = 'edit';
1541     if ($do == 'restore') $opened_tab = 'view';
1542
1543     media_tabs_details($image, $opened_tab);
1544
1545     echo '<div class="panelHeader"><h3>';
1546     [$ext] = mimetype($image, false);
1547     $class = preg_replace('/[^_\-a-z0-9]+/i', '_', $ext);
1548     $class = 'select mediafile mf_' . $class;
1549
1550     $attributes = $rev ? ['rev' => $rev] : [];
1551     $tabTitle = sprintf(
1552         '<strong><a href="%s" class="%s" title="%s">%s</a></strong>',
1553         ml($image, $attributes),
```

```
1554         $class,
1555         $lang['mediaview'],
1556         $image
1557     );
1558     if ($opened_tab === 'view' && $rev) {
1559         printf($lang['media_viewold'], $tabTitle, dformat($rev));
1560     } else {
1561         printf($lang['media_' . $opened_tab], $tabTitle);
1562     }
1563 
1564     echo '</h3></div>' . NL;
1565 
1566     echo '<div class="panelContent">' . NL;
1567 
1568     if ($opened_tab == 'view') {
1569         media_tab_view($image, $ns, null, $rev);
1570     } elseif ($opened_tab == 'edit' && !$removed) {
1571         media_tab_edit($image, $ns);
1572     } elseif ($opened_tab == 'history' && $conf['mediarevisions']) {
1573         media_tab_history($image, $ns);
1574     }
1575 
1576     echo '</div>' . NL;
1577 }
1578 
1579 /**
1580  * prints the namespace tree in the mediamanager popup
1581  *
1582  * Only allowed in mediamanager.php
1583  *
1584  * @author Andreas Gohr <andi@splitbrain.org>
1585  */
1586 function tpl_mediaTree()
1587 {
1588     global $NS;
1589     echo '<div id="media__tree">';
1590     media_nstree($NS);
1591     echo '</div>';
1592 }
1593 
1594 /**
1595  * Print a dropdown menu with all DokuWiki actions
1596  *
1597  * Note: this will not use any pretty URLs
1598  *
1599  * @param string $empty empty option label
1600  * @param string $button submit button label
1601  *
1602  * @author Andreas Gohr <andi@splitbrain.org>
1603  * @deprecated 2017-09-01 see devel:menus
1604  */
```

```
1605 function tpl_actiondropdown($empty = '', $button = '&gt;')
1606 {
1607     dbg_deprecated('see devel:menus');
1608     $menu = new MobileMenu();
1609     echo $menu->getDropdown($empty, $button);
1610 }
1611
1612 /**
1613  * Print a informational line about the used license
1614  *
1615  * @param string $img print image? (|button|badge)
1616  * @param bool $imgonly skip the textual description?
1617  * @param bool $return when true don't print, but return HTML
1618  * @param bool $wrap wrap in div with class="license"?
1619  * @return string
1620  *
1621  * @author Andreas Gohr <andi@splitbrain.org>
1622  */
1623 function tpl_license($img = 'badge', $imgonly = false, $return = false,
$wrap = true)
1624 {
1625     global $license;
1626     global $conf;
1627     global $lang;
1628     if (!$conf['license']) return '';
1629     if (!is_array($license[$conf['license']])) return '';
1630     $lic = $license[$conf['license']];
1631     $target = ($conf['target']['extern']) ? ' target="' .
$conf['target']['extern'] . '"' : '';
1632
1633     $out = '';
1634     if ($wrap) $out .= '<div class="license">';
1635     if ($img) {
1636         $src = license_img($img);
1637         if ($src) {
1638             $out .= '<a href="' . $lic['url'] . '" rel="license"' .
$target;
1639             $out .= '><img src="' . DOKU_BASE . $src . '" alt="' .
$lic['name'] . '" /></a>';
1640             if (!$imgonly) $out .= ' ';
1641         }
1642     }
1643     if (!$imgonly) {
1644         $out .= $lang['license'] . ' ';
1645         $out .= '<bdi><a href="' . $lic['url'] . '" rel="license"
class="urlextern"' . $target;
1646         $out .= '>' . $lic['name'] . '</a></bdi>';
1647     }
1648     if ($wrap) $out .= '</div>';
1649
```

```
1650     if ($return) return $out;
1651     echo $out;
1652     return '';
1653 }
1654
1655 /**
1656  * Includes the rendered HTML of a given page
1657  *
1658  * This function is useful to populate sidebars or similar features in
a
1659  * template
1660  *
1661  * @param string $pageid The page name you want to include
1662  * @param bool $print Should the content be printed or returned only
1663  * @param bool $propagate Search higher namespaces, too?
1664  * @param bool $useacl Include the page only if the ACLs check out?
1665  * @return bool|null|string
1666  */
1667 function tpl_include_page($pageid, $print = true, $propagate = false,
$useacl = true)
1668 {
1669     if ($propagate) {
1670         $pageid = page_findnearest($pageid, $useacl);
1671     } elseif ($useacl && auth_quickaclcheck($pageid) == AUTH_NONE) {
1672         return false;
1673     }
1674     if (!$pageid) return false;
1675
1676     global $TOC;
1677     $oldtoc = $TOC;
1678     $html = p_wiki_xhtml($pageid, '', false);
1679     $TOC = $oldtoc;
1680
1681     if ($print) echo $html;
1682     return $html;
1683 }
1684
1685 /**
1686  * Display the subscribe form
1687  *
1688  * @author Adrian Lang <lang@cosmocode.de>
1689  * @deprecated 2020-07-23
1690  */
1691 function tpl_subscribe()
1692 {
1693     dbg_deprecated(Subscribe::class . '::show()');
1694     (new Subscribe())->show();
1695 }
1696
1697 /**
1698  * Tries to send already created content right to the browser
```

```
1699  *
1700  * Wraps around ob_flush() and flush()
1701  *
1702  * @author Andreas Gohr <andi@splitbrain.org>
1703  */
1704 function tpl_flush()
1705 {
1706     if (ob_get_level() > 0) ob_flush();
1707     flush();
1708 }
1709
1710 /**
1711  * Tries to find a ressource file in the given locations.
1712  *
1713  * If a given location starts with a colon it is assumed to be a media
1714  * file, otherwise it is assumed to be relative to the current template
1715  *
1716  * @param string[] $search locations to look at
1717  * @param bool $abs if to use absolute URL
1718  * @param array    &$imginfo filled with getimagesize()
1719  * @param bool $fallback use fallback image if target isn't found or
return 'false' if potential
1720  *                                false result is required
1721  * @return string
1722  *
1723  * @author Andreas  Gohr <andi@splitbrain.org>
1724  */
1725 function tpl_getMediaFile($search, $abs = false, &$imginfo = null,
$fallback = true)
1726 {
1727     $img = '';
1728     $file = '';
1729     $ismedia = false;
1730     // loop through candidates until a match was found:
1731     foreach ($search as $img) {
1732         if (str_starts_with($img, ':')) {
1733             $file = mediaFN($img);
1734             $ismedia = true;
1735         } else {
1736             $file = tpl_incdir() . $img;
1737             $ismedia = false;
1738         }
1739
1740         if (file_exists($file)) break;
1741     }
1742
1743     // manage non existing target
1744     if (!file_exists($file)) {
1745         // give result for fallback image
1746         if ($fallback) {
```

```
1747             $file = DOKU_INC . 'lib/images/blank.gif';
1748             // stop process if false result is required (if $fallback
is false)
1749         } else {
1750             return false;
1751         }
1752     }
1753
1754     // fetch image data if requested
1755     if (!is_null($imginfo)) {
1756         $imginfo = getimagesize($file);
1757     }
1758
1759     // build URL
1760     if ($ismedia) {
1761         $url = ml($img, '', true, '', $abs);
1762     } else {
1763         $url = tpl_basedir() . $img;
1764         if ($abs) $url = DOKU_URL . substr($url, strlen(DOKU_REL));
1765     }
1766
1767     return $url;
1768 }
1769
1770 /**
1771  * PHP include a file
1772  *
1773  * either from the conf directory if it exists, otherwise use
1774  * file in the template's root directory.
1775  *
1776  * The function honours config cascade settings and looks for the given
1777  * file next to the ´main´ config files, in the order protected, local,
1778  * default.
1779  *
1780  * Note: no escaping or sanity checking is done here. Never pass user
input
1781  * to this function!
1782  *
1783  * @param string $file
1784  *
1785  * @author Andreas Gohr <andi@splitbrain.org>
1786  * @author Anika Henke <anika@selfthinker.org>
1787  */
1788 function tpl_includeFile($file)
1789 {
1790     global $config_cascade;
1791     foreach (['protected', 'local', 'default'] as $config_group) {
1792         if (empty($config_cascade['main'][$config_group])) continue;
1793         foreach ($config_cascade['main'][$config_group] as $conf_file)
{
1794             $dir = dirname($conf_file);
```

```
1795             if (file_exists("$dir/$file")) {
1796                 include("$dir/$file");
1797                 return;
1798             }
1799         }
1800     }
1801
1802     // still here? try the template dir
1803     $file = tpl_incdir() . $file;
1804     if (file_exists($file)) {
1805         include($file);
1806     }
1807 }
1808
1809 /**
1810  * Returns <link> tag for various icon types (favicon|mobile|generic)
1811  *
1812  * @param array $types - list of icon types to display
(favicon|mobile|generic)
1813  * @return string
1814  *
1815  * @author Anika Henke <anika@selfthinker.org>
1816  */
1817 function tpl_favicon($types = ['favicon'])
1818 {
1819
1820     $return = '';
1821
1822     foreach ($types as $type) {
1823         switch ($type) {
1824             case 'favicon':
1825                 $look = [':wiki:favicon.ico', ':favicon.ico',
'images/favicon.ico'];
1826                 $return .= '<link rel="shortcut icon" href="' .
tpl_getMediaFile($look) . '" />' . NL;
1827                 break;
1828             case 'mobile':
1829                 $look = [':wiki:apple-touch-icon.png', ':apple-touch-
icon.png', 'images/apple-touch-icon.png'];
1830                 $return .= '<link rel="apple-touch-icon" href="' .
tpl_getMediaFile($look) . '" />' . NL;
1831                 break;
1832             case 'generic':
1833                 // ideal world solution, which doesn't work in any
browser yet
1834                 $look = [':wiki:favicon.svg', ':favicon.svg',
'images/favicon.svg'];
1835                 $return .= '<link rel="icon" href="' .
tpl_getMediaFile($look) . '" type="image/svg+xml" />' . NL;
1836                 break;
```

```
1837         }
1838     }
1839
1840     return $return;
1841 }
1842
1843 /**
1844  * Prints full-screen media manager
1845  *
1846  * @author Kate Arzamastseva <pshns@ukr.net>
1847  */
1848 function tpl_media()
1849 {
1850     global $NS, $IMG, $JUMPTO, $REV, $lang, $fullscreen, $INPUT;
1851     $fullscreen = true;
1852     require_once DOKU_INC . 'lib/exe/mediamanager.php';
1853
1854     $rev = '';
1855     $image = cleanID($INPUT->str('image'));
1856     if (isset($IMG)) $image = $IMG;
1857     if (isset($JUMPTO)) $image = $JUMPTO;
1858     if (isset($REV) && !$JUMPTO) $rev = $REV;
1859
1860     echo '<div id="mediamanager__page">' . NL;
1861     echo '<h1>' . $lang['btn_media'] . '</h1>' . NL;
1862     html_msgarea();
1863
1864     echo '<div class="panel namespaces">' . NL;
1865     echo '<h2>' . $lang['namespaces'] . '</h2>' . NL;
1866     echo '<div class="panelHeader">';
1867     echo $lang['media_namespaces'];
1868     echo '</div>' . NL;
1869
1870     echo '<div class="panelContent" id="media__tree">' . NL;
1871     media_nstree($NS);
1872     echo '</div>' . NL;
1873     echo '</div>' . NL;
1874
1875     echo '<div class="panel filelist">' . NL;
1876     tpl_mediaFileList();
1877     echo '</div>' . NL;
1878
1879     echo '<div class="panel file">' . NL;
1880     echo '<h2 class="a11y">' . $lang['media_file'] . '</h2>' . NL;
1881     tpl_mediaFileDetails($image, $rev);
1882     echo '</div>' . NL;
1883
1884     echo '</div>' . NL;
1885 }
1886
1887 /**
```

```php
 * Return useful layout classes
 *
 * @return string
 *
 * @author Anika Henke <anika@selfthinker.org>
 */
function tpl_classes()
{
    global $ACT, $conf, $ID, $INFO;
    /** @var Input $INPUT */
    global $INPUT;

    $classes = [
        'dokuwiki',
        'mode_' . $ACT,
        'tpl_' . $conf['template'],
        $INPUT->server->bool('REMOTE_USER') ? 'loggedIn' : '',
        (isset($INFO['exists']) && $INFO['exists']) ? '' : 'notFound',
        ($ID == $conf['start']) ? 'home' : ''
    ];
    return implode(' ', $classes);
}

/**
 * Create event for tools menues
 *
 * @param string $toolsname name of menu
 * @param array $items
 * @param string $view e.g. 'main', 'detail', ...
 *
 * @author Anika Henke <anika@selfthinker.org>
 * @deprecated 2017-09-01 see devel:menus
 */
function tpl_toolsevent($toolsname, $items, $view = 'main')
{
    dbg_deprecated('see devel:menus');
    $data = ['view' => $view, 'items' => $items];

    $hook = 'TEMPLATE_' . strtoupper($toolsname) . '_DISPLAY';
    $evt = new Event($hook, $data);
    if ($evt->advise_before()) {
        foreach ($evt->data['items'] as $html) echo $html;
    }
    $evt->advise_after();
}

```