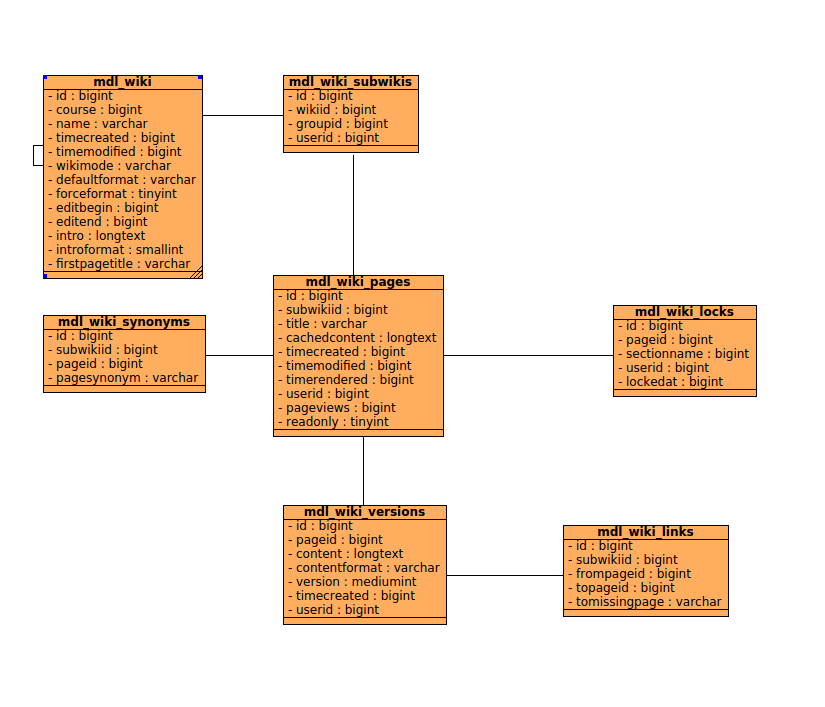
The wiki databases

**mdl\_wiki**

A row in the wiki table contains the information about an instance of a wiki

Description:

* id is a unique number for each wiki instance
* course is the course id of the course which the wiki is in
* name is the wiki’s name
* intro is the wiki’s description text
* introformat is the format of the description text which is always html so it is always
* timecreated is always 0
* timemodified is the unix time of the last edit
* firstpage title is the title of the first page of the wiki
* wikimode is either collaborative or individual
* defaultformat is the format of the wiki Html/creole/nwiki
* forceformat is Boolean 0/1 and is whether or not all pages in the wiki must fallow the default format
* editbegin and editend seem to always be 0

**mdl\_wiki\_subwiki**

A row in the subwiki table contains information about a subwiki if a wiki is collaborative there is only one sub wiki for that wiki but an individual wiki has a subwiki for every user with a page.

Description:

* id is a unique number for each subwiki
* wikiid is the id of the wiki which the subwiki belongs to
* groupid is the id of the group that the subwiki belongs to
* userid is the id of the user that the subwiki belongs to

**mdl\_wiki\_links**

Description:

* id is a unique number for each link
* subwikiid is the id of the subwiki where the page the link is on is found
* frompageid is the page id of the page with the link on it
* topageid is the id of the page the link goes to, it’s 0 if the page doesn’t exist
* tomissingpage is null if the linked page exist or the name of the page if the linked page doesn’t exist

**mdl\_wiki\_pages**

Description:

* id is a unique number for each page
* subwikiid is the id of the subwiki that the page belongs to
* title is the title of the page
* cachedcontent is the html for the latest version of the page
* timecreated is the unix time when the page was created
* timemodified is the unix time when the page was last modified
* timerendered is the unix time when the page was last viewed and the content was cached
* userid is the id of the last user who edited the page
* pageviews is the number of times a page has been viewed
* readonly is a Boolean 1/0 if the page is readonly or can be edited

**mdl\_wiki\_locks**

The locks table contain which pages are locked. A page or section becomes locked when another user is editing it.

Description:

* id is a unique id for each lock
* pageid is the id of the page that is locked
* sectionname null if the entire page is locked or the name of the section that is locked
* userid the id of the user who locked the page
* lockedat is the unix time at which the page was locked

**mdl\_wiki\_synonyms**

Description:

* id unique id for each synonym
* subwikiid id of the subwiki the synonym belongs to
* pageid id of the original page
* pagesynonym page name synonym

**mdl\_wiki\_versions**

Contains information of every version of a page whenever a page is edited a new version is created with the new changes.

Description:

* id unique number for each version
* pageid the id of the page
* content the content of the version
* contentformat is the format that the content is in creole/nwiki/html
* version is the version number of the page ex first version=1 second =2 etc..
* timecreated is the unix time when the version was created
* userid is the id of the user who created the version

How wiki works

**Wiki Files**

The most important functions and classes for the wiki are found in locallib.php, pagelib.php, and parser/parser.php. These files are where most of the real work gets done.

**lang/en/wiki.php**

wiki's language file.When get\_string() is called, it looks to this file to find the correct string.

**parser/markups/creole.php,nwiki.php,html.php**

These files contain the parsers for specific markups. Never used directly outside of the parsers folder.

**parser/markups/wikimarkup.php**

Contains the abstract parser that all other parsers extend. Not used outside of the parsers folder.

**parser/utils.php**

File for class parser\_utils. This class contains utility functions for parsers to use, and default callback functions for the parsers.

**parser/parser.php**

Classes: wiki\_parser\_proxy, generic\_parser

class wiki\_parser\_proxy

Eveything outside of the parser folder uses wiki\_parser\_proxy to interact with the parsers, through its static methods This way, other functions do not need to worry about figuring out which parser type to use, and keep track of. The parser\_proxy selects the right parser, and calls the necessary methods to parse a page's content.

* parse(&$string, $type, $options = array())
  + parses a given string of the markup *type.* Returns the parsed content
* get\_token($name, $type)
  + gets the token that represents *name* for a specific markup *type*
* get\_section(&$string, $type, $section, $all\_content = false)

**module.js**

Contains helper functions for the wiki module; init(), renew\_lock(), history(), deleteversion(), init\_tree(). These are not necessary for wiki to function properly, although they greatly enhance some functionality (history, the map tree, etc).

**files ending in \_form.php**

Files that end in \_form.php all contain classes that define the layout of the forms for other pages. These are used by their respective *page\_wiki\_?* Classes. For example, create\_form.php is used by *page\_wiki\_create* to display the form for creating a page.

**restoreversion.php**

This page is displayed when the user clicks on the restore button next to a page in the history table. This displays a confirmation page (*page\_wiki\_confirmrestore)* if a $confirm parameter was supplied, otherwise it creates a *page\_wiki\_restoreversion*, which restores the page.

**editcomments.php**

This is the page the user is directed to when they attempt to edit a comment. This page displays the edit comments form using *page\_wiki\_editcomment*

**renderer.php**

Contains the class that overrides the default renderer, adding functions to the renderer that are specific to this function

# Classes: mod\_wiki\_renderer, wiki\_files\_tree

Class mod\_wiki\_renderer

* page\_index()
  + outputs the box containing an index of all wiki instances in a course
* search\_result($records, $subwiki)
  + outputs a container that displays search results contained in records
* diff($pageid, $old, $new, $options = array())
  + Outputs the diff page, showing the differences between versions of a page. Highlights the differences
* diff\_paging\_bar($minpage, $maxpage, $page, $baseurl, $pagevar = 'page', $nocurr = false)
  + displays a small “bar” that provides quick access to other versions of the page that the user might want to compare with the current version
* wiki\_info()
  + outputs a container with the wiki's description inside
* tabs($page, $tabitems, $options)
  + outputs a tab bar, filled with $tabitems
* prettyview\_link($page)
  + outputs a link to prettyview.php, the printer-friendly version link
* wiki\_print\_subwiki\_selector($wiki, $subwiki, $page, $pagetype = 'view')
  + outputs the subwiki selector. This is populated with either groups or users, depending on the wiki mode. If the wiki mode is collaborative, nothing gets output at all
* menu\_map($pageid, $currentselect)
  + outputs the dropdown list of options for the wiki map view
* wiki\_files\_tree($context, $subwiki)
  + outputs a new wiki\_files\_tree
* render\_wiki\_files\_tree(wiki\_files\_tree $tree)
  + outputs the wiki file tree. This function initializes a javascript module that is used for the tree view
* menu\_admin($pageid, $currentselect)
  + outputs the dropdown list of administration options, for use in admin.php

**prettyview.php**

This page creates the printer-friendly version of a page. Uses *page\_wiki\_prettyview*.

**overridelocks.php**

This page is used exclusively when attempting to edit a page. If a user has permissions to override locks on pages, a button appears when they attempt to edit a locked page. Pressing this button calls overridelocks.php to override the lock on the page.

**map.php**

This page creates the map view for the wiki, using *page\_wiki\_map*.

**lock.php**

Creates a page\_wiki\_lock. Used to renew the lock on a page. This file is used to respond to ajax calls when a user is on edit.php. When someone is editing a page, every 25 an ajax request gets sent to this file, to renew the lock on the page.

**lib.php**

Overrides moodle functions such as ::addinstance for the wiki module. Not used from inside this module.

**instancecomments.php**

This is the page that gets displayed when comments are added or deleted. This creates the form that is appropriate for the action that the user wants to accomplish. If there is no action, it is assumed that user wants to delete the comment. Objects that can be created depending on the case are:

*page\_wiki\_handlecomments*

*page\_wiki\_deletecomments*

*mod\_wiki\_comments\_form*

**index.php**

This file displays a list of all the instances of wiki in a course. The required parameter, id, is the id of the course who's instances will be displayed. This page echoes html directly, and using the html\_writer provided by moodle.

**filesedit.php**

This is the file responsible for deleting and uploading new files to the wiki. It prepares a standard file manager (with a drag and drop option if javascript is enabled), using *mod\_wiki\_filesedit\_form*.

**files.php**

The page that is displayed when the files tab is clicked. This file is responsible for managing the files that have been uploaded to this wiki. Unlike most other files, this uses the renderr directly to output a subwiki selector, and a *wiki\_files\_tree* which can be found in renderer.php. The manage files button takes the user to filesedit.php

**diff.php**

This is the page displayed whenthe compare button is pressed. Required parameters are pageid, compare, and comparewith. compare and comparewith are version ids. Creates a page\_wiki\_diff, which creates the visuals for the page.

**comments.php**

This page displays all of the comments associated with a wiki page. Comments are stored in the *comment* table, they are not specific to the wiki. Creates a *page\_wiki\_comments*, which does the work to display the comments to the user.

**admin.php**

This is the wiki's administration page. If a parameter *delete* is supplied, the page id in *delete* will be removed from the wiki. The user is then directed to create that page. If *toversion* and *fromversion* are provided, all versions of a page between *fromversion* and *toversion* will be deleted from the wiki. Creates a *page\_wiki\_admin* that display the administration options to the user.

**view.php**

Viewing Pages

There are 3 possible ways to view a page in a wiki, and a different action is required in each case:

1. Coming from a course→show the first page
2. Request to view a specific page→show the page that was requested
3. Request to see a specific group or user's wiki page. This most often happens when using individual wiki mode, and the user picks another user from the subwiki selector.

Case 1: When you come from a course

When you click on the link to go to the wiki, you are sent to view.php. The ID parameter is the wiki module's ID in the table mdl\_course\_modules. This table keeps a record of every module created, with the course it is in, the module type, etc. Then *wiki\_get\_wiki* (locallib.php) is called, with the instance ID of the wiki. This function returns the row from the database *wiki* associated with this instance ID, using DB->get\_record. Then it gets the current group ID (this is some default value if groups are not enabled), the it does a *get\_subwiki\_by\_group* (locallib.php). This returns the row from the *wiki\_subwikis* table that is associated with the group and user. If the wiki is in collaborative mode, the user field is always 0. For individual mode, the user field is the ID of the current user (ID from *user* table). If the subwiki doesn't exists you get redirected to create.php. Then, view attempts to get the first page of the subwiki by calling *wiki\_get\_first\_page* with the subwiki's ID. All pages seem to be in subwikis, they are not associated directly with a wiki. *wiki\_get\_first\_page* gets the first page of the wiki.

Case 2:When you request to see a specific page(you can get here from a variety of places)

This will be the case whenever view.php is run with a pageid parameter. The $page variable is set by calling wiki\_get\_page with the pageid. Simplest case.

Case 3: When you try to see someone else's wiki page

This is the case if view.php is run with the wid and title parameters. The page gets the wiki associated with wid, the course instances, etc. Then it sets group ids based on what type of wiki is being used. Then it checks permissions, and calls *wiki\_get\_page\_by\_title* with the page's title and subwiki id, which retrieves the page from *wiki\_pages* using DB->get\_record.

Interesting functions used here: *wiki\_print\_subwiki\_selector*, found in renderer.php.

All Cases

After $page has been set by one of the above cases, view.php makes a new *page\_wiki\_view* (pagelib.php).Methods of this class are called to display the header, content, and footer. This involves calling *wiki\_print\_page\_content*, which either prints the cached content, or if enough time has passed to refresh the cache, refreshes the cache(the refresh time is 30 seconds). Then uses the

$page->cachedcontent to get the contents of the page.

**create.php**

Creating new wiki pages

There are 2 actions that this file can perform:

1. When the action is “new”, provides a form for adding a title, format, etc.
2. When the action is “create”, don't display anything, just create the page in the database and redirect to the edit page.

This file must be run with an action parameter, and either a subwiki ID, or wiki ID, group ID, and user ID.

All actions: get the wiki and subwiki that go with the parameters.

Action 1: When the user wants to make a new page and the title, format are not known. This normally happens when the “new” link is clicked.

If the page's title is known and the “force format” option is selected, we know enough information to go straight through to the edit page, after creating the new page. Otherwise, display the create page form. The create page form then gets submitted back to create.php, but with the “create” action instead. See Action 2.

Action 2: When a new page needs to be created, but the title and format can be assumed. This can happen in several places, like when a page is created by using square brackets in another wiki page.

Create a new *page\_wiki\_create*, and use this to create a new page with the specified title. Then, redirect to edit.php to edit the new page.

**viewversion.php**

Required parameters are pageid and versionid. The page ID is an id from *wiki\_pages* and the version ID is an ID from *wiki\_versions*. This file creates a new *page\_wiki\_viewversions*(pagelib.php), sets the page, sets the version ID, and displays the eader, footer, and content. Most of the real work happens in page\_wiki\_viewversion. This displays a form that is essentially the same as in view.php, but with options to restore this version of the page, showing the version number, the author, and time modified.

**history.php**

Required parameter is a page ID. This file gets the subwiki, wiki, and page using functions from locallib.php, the passes them to a new page\_wiki\_history. This builds and displays the history of the page, and will probably need to be mostly rewritten for socialwiki, because the version tree is very different.

**search.php**

Not linked to directly, but accessed via the search box and button. Creates a *page\_wiki\_search,* which takes care of all of the actual searching.

**locallib.php**

Quick description of the purpose of each function. Any custom SQL statement (used with get\_record\_sql) from these functions is also included in the description.

*• wiki\_get\_wiki($wikiid)*

Returns the row from *wiki* that goes with wikiid, as an stdClass object

*• wiki\_get\_subwikis($wikiid)*

Returns the rows from *wiki\_subwikis* that are part of the wiki with wikiid as an array

*• wiki\_get\_subwiki\_by\_group($wikiid, $groupid, $userid = 0)*

Returns the row from *wiki\_subwiki* sthat belongs to the specified wiki, with the correct groupid and userid. UserID defaults to zero because a userID is not necessary to make a group subwiki.

*• wiki\_get\_subwiki($subwikiid)*

Returns the row from *wiki\_subwikis* that goes with subwikiid

*• wiki\_add\_subwiki($wikiid, $groupid, $userid=0)*

Adds a new row to *wiki\_subwikis* with the information in the parameters. Returns the ID that is returned by insert\_record

*• wiki\_get\_wiki\_from\_pageid($pageid)*

Returns the wiki that the page specified by pageid belongs to.

"SELECT w.\*

FROM {wiki} w, {wiki\_subwikis} s, {wiki\_pages} p

WHERE p.id = ? AND

p.subwikiid = s.id AND

s.wikiid = w.id";

*• wiki\_get\_page($pageid)*

Returns the page that has the specified pageid

*• wiki\_get\_current\_version($pageid)*

Returns the current version of the page specified by pageid.

"SELECT \*

FROM {wiki\_versions}

WHERE pageid = ?

ORDER BY version DESC";

*• wiki\_get\_last\_version($pageid)*

Exactly the same thing as *wiki\_get\_current\_version*

*• wiki\_get\_section\_page($page, $section)*

Returns the content of the section specified by $section on the page $page. This gets the current version of the page and then uses *wiki\_parser\_proxy* to get the section's content.

*• wiki\_get\_page\_by\_title($swid, $title)*

Returns the page from subwiki *swid* with title *title*

*• wiki\_get\_version($versionid)*

Returns the page version with *versionid*

*• wiki\_get\_first\_page($subwikiid, module)*

*module* is an unused parameter. Returns the first page of the subwiki with *subwikiid*.

"SELECT p.\*

FROM {wiki} w, {wiki\_subwikis} s, {wiki\_pages} p

WHERE s.id = ? AND

s.wikiid = w.id AND

w.firstpagetitle = p.title AND

p.subwikiid = s.id";

*• wiki\_save\_section($wikipage, $sectiontitle, $sectioncontent, $userid)*

Creates a new version of the page *wikipage* with the section *sectiontitle* updated with new content. This uses *wiki\_parser\_proxy* to get the section. Also calls wiki\_save\_page.

*• wiki\_save\_page($wikipage, $newcontent, $userid)*

Inserts a new row into *wiki\_versions* with the new content for the page, and calls refresh\_cachedcontent to set the initial cache for the page.

*• wiki\_refresh\_cachedcontent($page, $newcontent=null)*

Updates the field *cachedcontent* in *wiki\_pages* that is associated with this page. Parses the most recent version of the page, and then caches the result. This cuts down on the time needed to view a page.

*• wiki\_restore\_page($wikipage, $newcontent, $userid)*

Saves *newcontent* to *wikipage*. Does the same thing as wiki\_save\_page, but returns the page after it has been updated.

*• wiki\_refresh\_page\_links($page, $links)*

Removes all records from *wiki\_links* that originate from *page*. Then, adds each link in *link* to *wiki\_links*.

*• wiki\_create\_page($swid, $title, $format, $userid)*

This function checks capabilities before doing anything. Then it adds a page with the proper information to *wiki\_pages* and a new version to *wiki\_versions*. This function does not set the content of this page. This function causes the *cachedcontent* of the page to expire, forcing an update of the cache before the page is viewed.

*• wiki\_make\_cache\_expire($pagename)*

Causes the cachedcontent of the page to expire, forcing a cache update before the page can be viewed again.

"UPDATE {wiki\_pages}

SET timerendered = 0

WHERE id IN ( SELECT l.frompageid

FROM {wiki\_links} l

WHERE l.tomissingpage = ?

)";

*• wiki\_get\_wiki\_page\_version($pageid, $version)*

Returns the version *version* of the specified page.

*• wiki\_count\_wiki\_page\_versions($pageid)*

Returns the number of versions of a page

*• wiki\_get\_linked\_to\_pages($pageid)*

Returns all the *pageid*s that this page links to

*• wiki\_get\_linked\_from\_pages($pageid)*

Returns all the *pageid*s that link to this pag

*• wiki\_get\_contributions($swid, $userid)*

Returns all the versions of pages from subwiki *swid* that the specified user has created.

"SELECT v.\*

FROM {wiki\_versions} v, {wiki\_pages} p

WHERE p.subwikiid = ? AND

v.pageid = p.id AND

v.userid = ?";

*• wiki\_get\_missing\_or\_empty\_pages(swid)*

Returns the title, id, and subwikiids of pages that are empty or missing. A page is empty if there is only one version of the page (the initial empty version), and it is missing if it is linked to by another page and does not yet exist.

"SELECT DISTINCT p.title, p.id, p.subwikiid

FROM {wiki} w, {wiki\_subwikis} s, {wiki\_pages} p

WHERE s.wikiid = w.id and

s.id = ? and

w.firstpagetitle != p.title and

p.subwikiid = ? and

1 = (SELECT count(\*)

FROM {wiki\_versions} v

WHERE v.pageid = p.id)

UNION

SELECT DISTINCT l.tomissingpage as title, 0 as id, l.subwikiid

FROM {wiki\_links} l

WHERE l.subwikiid = ? and

l.topageid = 0";

*• wiki\_get\_page\_list($swid)*

Returns an array of all the pages in subwiki *swid*

*• wiki\_get\_orphaned\_pages($swid)*

Returns a list of pages in the subwiki *swid* that are not linked to by any other page.

SELECT p.id, p.title

FROM {wiki\_pages} p, {wiki} w , {wiki\_subwikis} s

WHERE p.subwikiid = ?

AND s.id = ?

AND w.id = s.wikiid

AND p.title != w.firstpagetitle

AND p.id NOT IN (SELECT topageid FROM {wiki\_links} WHERE subwikiid = ?)";

*• wiki\_search\_title($swid, $search)*

Returns any pages in subwiki *swid* that have a title LIKE *search*

*• wiki\_search\_all($swid, $search)*

Returns any pages in subwiki *swid* that contain *search* in either *cachedcontent* or *title*

*• wiki\_get\_user\_info($userid)*

Returns the row in *user* that holds information about the user with id *userid*

*• wiki\_increment\_pageviews($page)*

Increments the page's pageviews. Updates *wiki\_pages*

*• wiki\_get\_formats()*

Returns a list of hard-coded supported formats

*wiki\_parse\_content($markup, $pagecontent, $options = array())*

Uses *wiki\_parser\_proxy* to parse the page's content for the specified markup. Returns the content after parsing.

*• wiki\_parser\_link($link, $options=null)*

A callback function for the parser when it comes across a wiki link. Returns the information that a parser needs in order to display internal links

*• wiki\_parser\_table($table)*

A callback function for the parsers when they encounter a table. Returns the complete html necessary to display a table.

*• wiki\_parser\_real\_path($url, $context, $component, $filearea, $swid)*

Returns a url to an absolute path on the server.

*• wiki\_parser\_get\_token($markup, $name)*

Returns the token for *name* in the specified markup language. Just calls *wiki\_parser\_proxy::get\_token*

*• wiki\_user\_can\_view($subwiki)*

Returns a boolean indicating whether or not the current user is allowed to view the specified subwiki. The result depends on the wiki options and permissions.

*• wiki\_user\_can\_edit(subwiki)*

Returns a boolena indicating whether or not the current user is allowed to edit pages in the specified subwiki.

*• wiki\_is\_page\_section\_locked($pageod, $userid, $section = null)*

Checks to see if a lock exists in *wiki\_locks* for the specified page, userid, and section

*• wiki\_set\_lock($pageid, $userid, $section = null, $insert = false)*

Adds a new lock to the page *pageid* , and specifi section if *section* is not null.

*• wiki\_delete\_locks($pageid, $uesrid = null, $section = null, $delete\_from\_db = true, $delete\_section\_and\_page = false)*

Deletes unused locks. Locks should be deleted after saving changes, for example, when they are no longer necessary.

*• wiki\_delete\_old\_locks()*

Deletes any locks that have been around for longer than an hour.

*• wiki\_delete\_links($linkid = null, $topageid = null, $frompageid = null, $subwikiid = null)*

Deletes any links in *wiki\_links* that match any of the criteria specified by the parameters

*• wiki\_delete\_synonym($subwikiid, $pageid = null)*

Deletes synonyms from *wiki\_synonyms* that are related to the page or subwiki that are specified in the parameters.

*• wiki\_delete\_pages($context, $pageids = null, $subwikiid = null)*

Deletes all of the pages specified in *pageids* from *wiki\_pages*

*• wiki\_delete\_page\_versions($deleteversions)*

Remove all the versions specified by ids in *deleteversions.* If deleteversions is zero all versions of the page will be deleted. Does not delete any pages in *wiki\_pages*

*• wiki\_get\_comment($commentid)*

Returns the comment with *commentid*

*• wiki\_get\_comments($contextid, $pageid)*

Returns an array of all of the comments for this page and context

*• wiki\_add\_comment($context, $pageid, $content, $editor)*

Adds a single comment to the database for the page specified.

*• wiki\_delete\_comment($idcomment, $context, $pageid)*

Deletes the comment with *idcomment* from the page *pageid*

*• wiki\_delete\_comments\_wiki()*

Gets the course module and deletes all of the comments from wiki with the current contextid.

*• wiki\_add\_progress($pageid, $oldversionid, $versionid, $progress)*

Forget about this function

*• wiki\_get\_wiki\_page\_id($pageid, $id)*

Returns the version with the field id in *wiki\_versions* equal to *id.*

*• wiki\_print\_page\_content($page, $context, $subwikiid)*

If WIKI\_REFRESH\_CACHE\_TIME has passed since this page was last cached, updates the cache for this page. Then, uses OUTPUT to echo html for a box containing this page's content to the webpage. Also increments the pageviews for this page.

*• wiki\_trim\_string($text, $limit = 25)*

Returns a string with maximum length of *limit* characters, caracters are cut from the end If the string is longer that *limit*

*• wiki\_print\_edit\_form\_default\_fields($format, $pageid, $version = • 1, $upload = false, $deleteuploads = array())*

This function directly outputs several html elements. Outputs the default edit form fields and buttons.

*• wiki\_print\_upload\_table($context, $filearea, $fileitemid, $deleteuploads = array())*

Directly outputs html. Prints out a table that shows a list of the files that are attached to the wiki page.

*• wiki\_build\_tree($page, $node, &$keys)*

Builds a tree for a wiki page. Recursive. *Node* is the starting node. This is not a version tree, it is used in *page\_wiki\_map* for one of the map views.

*• wiki\_get\_linked\_pages($pagid)*

Returns an array of pages that the page *pageid* links to.

"SELECT p.id, p.title

FROM {wiki\_pages} p

JOIN {wiki\_links} l ON l.topageid = p.id

WHERE l.frompageid = ?

ORDER BY p.title ASC";

*• wiki\_get\_updated\_pages\_by\_subwiki($swid)*

Retuns a list of pages, in descending order by timemodified, of pages that have been modified since the user logged in last

"SELECT \*

FROM {wiki\_pages}

WHERE subwikiid = ? AND timemodified > ?

ORDER BY timemodified DESC";

Editing a page

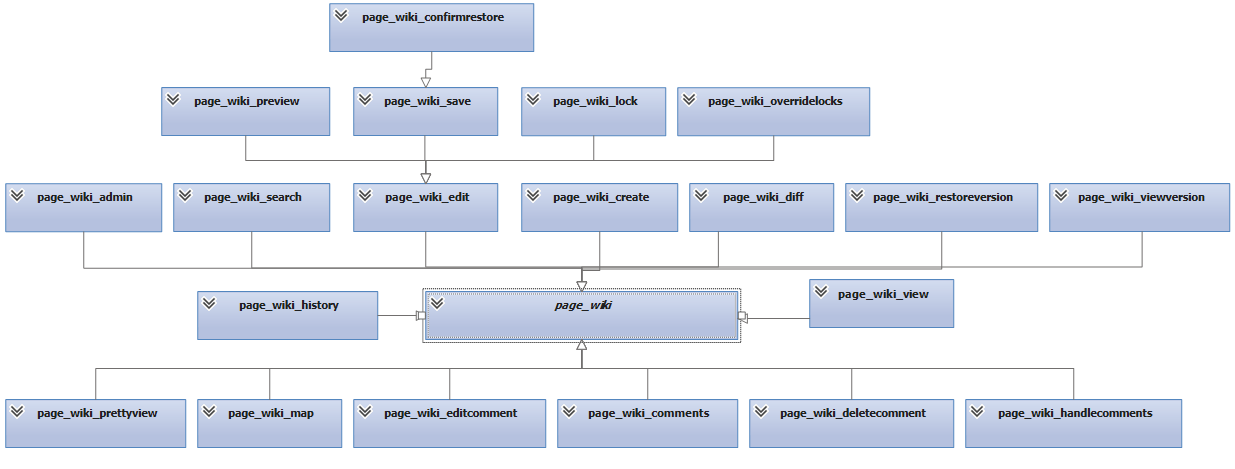
When the edit tab is clicked the pageid is sent to edit.php. In edit.php wiki\_get\_page is called passing the pageid as an argument. wiki\_get\_page searches the wiki\_pages table in the database for the page id and returns the row. Edit.php stores this in $page. wiki\_get\_subwiki is then called passing the subwikiid from $page as an argument. wiki\_get\_subwiki searches the wiki\_subwiki table for the subwikiid and returns that row. Edit.php stores this in $subwiki. wiki\_get\_wiki is then called with the wikiid from $subwiki passed as the argument. wiki\_get\_wiki searches the wiki table for the wikiid and returns that row edit.php stores this in $wiki. And then $wikipage = new page\_wiki\_edit then $wikipage->print\_content() is called which calls print\_edit() that calls wiki\_get\_current\_version with $this->page->id as an argument. wiki\_get\_current\_version searches the wiki\_versions table for all entries with the pageid it then orders them in descending order and returns the first element of the array. This is stored in $version the contents of $version are then displayed in the editor.

When the edit link next to a header is pressed the pageid and the section are sent to edit.php the page wiki and subwiki are setup same as when the edit tab is clicked. This time because there is a section wiki\_get\_section\_page gets called passing $page and $section as arguments. wiki\_get\_section\_page calls wiki\_get\_current\_version passing $ page->id, the return is stored in $version. Then wiki\_parser\_proxy::get\_section is called pasing $version->content, $version->contentformat and $section as arguments get\_section creates a parser instance for whatever format the wiki is in. The it calls get\_section() on that instance which uses a regular expression to find the headers and get all content that pertains to that section of the wiki this is then returned and stored in edit.php as $sectioncontent.

If the save button on the edit page is pressed edit.php is called and $option is ‘save’ all the page wiki and sub wiki is retrieved same as above. A new page\_wiki\_save is created as $wikipage. Then print\_content is called on $wikipage. This then calls print\_save (). The data from the edit\_form is the retrived using get\_data. Then wiki\_page\_save is called passing $this->page, $data->newcontent and $USER->id. Then wiki\_get\_current\_version is called and the return is stored in $version the content of $version is then set to $newcontent and $version->version is increased.Then the new $version is inserted into the wiki\_versions database. wiki\_refresh\_cachedcontent is then called the new content is then parsed and the wiki\_pages database is updated with the new parsed content as cashedcontent. The locks are then removed from the wiki\_lock databse and the page redirects to view.php

If preview is clicked new page\_wiki\_preview is created print\_content is called this calls print\_preview which parses the content and displays it on the page

**Pagelib**



**Classes**

[**page\_wiki**](#page_wiki)

[**page\_wiki\_admin**](#page_wiki_admin)

[**page\_wiki\_comments**](#page_wiki_comments)

[**page\_wiki\_confirmrestore**](#page_wiki_confirmrestore)

[**page\_wiki\_create**](#page_wiki_create)

[**page\_wiki\_deletecomment**](#page_wiki_deletecomment)

[**page\_wiki\_diff**](#page_wiki_diff)

[**page\_wiki\_edit**](#page_wiki_edit)

[**page\_wiki\_editcomment**](#page_wiki_editcomment)

[**page\_wiki\_handlecomments**](#page_wiki_handlecomments)

[**page\_wiki\_history**](#page_wiki_history)

[**page\_wiki\_lock**](#page_wiki_lock)

[**page\_wiki\_map**](#page_wiki_map)

[**page\_wiki\_overridelocks**](#page_wiki_overridelocks)

[**page\_wiki\_prettyview**](#page_wiki_prettyview)

[**page\_wiki\_preview**](#page_wiki_preview)

[**page\_wiki\_restoreversion**](#page_wiki_restoreversion)

[**page\_wiki\_save**](#page_wiki_save)

[**page\_wiki\_search**](#page_wiki_search)

[**page\_wiki\_view**](#page_wiki_view)

* **page\_wiki**

page wiki is an abstract class that every other page class inherits from

Attributes:

* + **gid** is the group id number
  + **modcontext** is the module context object
  + **page** is the current page
  + **subwiki** is the subwiki the page is found in
  + **tabs** is an array of all the tabs
  + **tabs\_options** is an array of tab options
  + **title** is the title of the page
  + **uid** is the current user’s id
  + **wikioutput** is a wiki renderer object

methods:

* + create\_navbar
    - creates the common items of the navbar in every page type
  + print\_content
    - the function must be overwritten to print page content
  + print\_footer
    - prints the page footer
  + print\_header
    - prints the page header
  + set\_gid
    - sets the group id for the page
  + set\_page
    - takes parameter page which is the current page
    - sets page to the parameter passed in
    - sets title of page to the current page’s title
    - calls PAGE->set\_title passing title
  + set\_title
    - sets page to null and sets title to the current page title
    - calls PAGE->set\_title passing title
  + set\_uid
    - sets the current user’s id as the uid for the page
  + set\_url
    - needs to be overwritten by every type of page
  + \_\_construct
    - Sets subwiki to the current subwiki
    - Sets modcontext to context\_module::instance($PAGE->cm->id);
    - Sets wikioutput to the wiki renderer
    - Sets up the PAGE global object
  + Print\_pagetitle
    - Prints out the current page title
  + Process\_session\_url
    - Deletes locks if the page is edit
  + Setup\_tabs
    - Sets up which tabs are displayed on the page by unsetting the tabs that the user doesn’t have the capability to see.
  + Set\_session\_url
    - Unsets SESSION->wikipreviousurl
* **page\_wiki\_view**

Attributes:

* + **Coursemodule** is the course module id

Methods:

* print\_header
  + calls parent print header and also prints the subwiki selector by calling wiki\_print\_subwiki\_selector
  + calls print\_pagetitle
* print\_content
  + checks that the user can view the subwiki
  + calls wiki\_print\_page\_content to print the page content
* set\_url
  + sets the page url
* set\_coursemodule
  + sets the coursemodule to the current pages course module id
* create\_navbar
  + sets the PAGE navbar
* **page\_wiki\_edit**

Attributes:

* + **section** the string name of the section to be edited
  + **overridelock** Boolean as to whether or not the lock can be over written. Default false
  + **versionnumber** the version number of the version being edited. Default -1
  + **upload** default false
  + **attachements** default 0
  + **deleteuploads**is an array
  + **format** is the format of the wiki

Methodes:

* \_\_construct (wiki,subwiki,cm)
  + Calls parent construct and adds prints the titleof the page/ section as a heading
* print\_header
  + calls parent print\_header and adds the title
* print\_content
  + checks if the user can edit and then calls [print\_edit](#print_edit)
* set\_url
  + sets the pages url
* set\_session\_url
  + sets the SESSION-> wikipreviousurl
* set\_section (sectioncontent, section)
  + sets the page’s sectioncontent to sectioncontent and the page’s section to section
* set\_versionnumber (versionnumber)
  + sets the page’s versionnumber to versionnumber
* set\_overridelock (override)
  + sets the page’s overridelock to override
* set\_upload (upload)
  + sets the page’s upload to upload
* set\_attachements (attachements)
  + sets the page’s attachments to attachments
* set\_deleteuploads (deleteuploads)
  + sets the page’s deleteuploads to deleteuploads
* create\_navbar
  + sets up PAGE->navbar
* check\_locks
  + creates a button if user is editing page and someone with override privileges wants to edit the page and allows the user with override privileges to edit that page.
* print\_edit (content=null)
  + gets the version contents from the database, creates a new mod\_wiki\_edit\_form then sets the data of the form to the page data and contents of the version.
* **page\_wiki\_comments**

Methods:

* + print\_header
    - calls parent print\_header and prints the title
  + print\_content
    - displays a table of comments with user info for the wiki
  + set\_url
    - sets the page’s url
  + create\_navbar
    - sets PAGE->navbar
* **page\_wiki\_editcomment**

Attributes :

* + **comment** is the comment’s row from the mdl\_comments table
  + **action** is a string of the action the user wishes to do
  + **form** is a mod\_wiki\_comments\_form
  + **format** the format of the wiki

Methods:

* set\_url
  + sets the page’s url
* print\_header
  + calls parent print\_header and prints page title
* print\_content
  + calls [add\_comment\_form](#add_comment_form) or [edit\_comment\_form](#edit_comment_form) depending on the action
* set\_action(action, comment)
  + set the pages action and comment and if the format is html creates a new mod\_wiki\_comments\_form
* create\_navbar
  + sets PAGE->navbar
* setup\_tabs
  + calls parent setup\_tabs with comments as the active tab
* add\_comment\_form
  + if the format is html
    - creates a new stdClass **com**
    - then sets the form data to **com**
    - displays the form
  + else it calls wiki\_print\_editor\_wiki
* edit\_comment\_form(com)
  + if html setsup com then sets the form data to com and displays the form
  + else calls wiki\_print\_editor\_wiki
* **page\_wiki\_search**

Attributes:

* **search\_result** is an array of pages that match a search

Methods:

* create\_navbar
  + sets the PAGE-> navbar
* set\_search\_string (search,searchcontent)
  + if searchcontent is 1 it calls wiki\_search\_all and searches for all content from a page including title
  + else it calls wiki\_search\_title and only searches for the title
* set\_url
  + sets the page’s url
* print\_content
  + outputs the search results by calling search\_result from the renderer
* **page\_wiki\_create**

Attributes:

* **format** the wikis format
* **swid** the subwiki id
* **wid** the wiki id
* **action** a string either new or create
* **mform** a mod\_wiki\_create\_form
* **groups** the groups who have access to the page

Methods:

* print\_header
  + calls parent print\_header
* set\_url
  + sets the page’s url
* set\_format
  + sets the page’s format
* set\_wid
  + sets the page’s wiki id
* set\_swid
  + sets the page’s subwiki id
* set\_availablegroups
  + sets the page’s groups
* set\_action
  + sets the page’s action and creates sets **mform** to a new mod\_wiki\_create\_form
* create\_navbar
  + sets PAGE->navbar
* print\_content
  + sets the data for **mform** and then displays it
* create\_page
  + gets the data from **mform**
  + creates a new page using wiki\_create\_page
  + returns the id of the page that was created
* **page\_wiki\_preview**

Attributes:

* **newcontent** is the content from the editor

Methods:

* \_\_construct
  + Calls parent \_\_construct and adds an update button
* Print\_header
  + Calls parent print\_header
* print\_content
  + calls print\_preview if the user has the capability to edit the page
* set\_new\_content
  + sets  **newcontent**
* set\_url
  + sets the page’s url
* check\_locks
  + returns true
* print\_preview
  + gets the current version of the page
  + creates a new mod\_wiki\_edit\_form
  + gets the content from the forms editor parses the content and displays it using [print\_edit](#print_edit)
* **page\_wiki\_diff**

Attributes:

* **compare** the version number of the older page
* **comparewith** the version number of the newerpage

Methods:

* print\_header
  + calls parent print\_header and prints out a heading with the names of the 2 versions being compared
* set\_url
  + sets the page’s url
* set\_comparison
  + sets **compare** and **comparewith** to the two versions to be compared
* setup\_tabs
  + sets up the tabs with history being the active tab
* print\_diff\_content
  + gets the total number of versions
  + gets the two versions to be compared from the database
  + uses ouwiki\_diff\_html to highlight the diffrences between the two versions and prints the two versions with the diffrences highlighted
* **page\_wiki\_history**

Attributes:

* **paging** the current page
* **rowsperpage**  Items per page
* **allversion** if **allversion** != 0, all versions will be printed in a signle table

Methods:

* \_\_construct
  + Calls parent \_\_construct and calls js inint for M.mod\_wiki.history
* print\_header
  + calls parent print\_header and prints the page title
* print\_pagetitle
  + prints the page title
* print\_content
  + checks capability to view page and calls print\_history\_content
* set\_url
  + sets the page’s url
* set\_paging
  + sets **paging**
* set\_allversion
  + sets **allversion**
* create\_navbar
  + set PAGE->navbar
* print\_history\_content
  + gets all the versions of a page and displays them in a table with radio buttons to select versions to compare and the user name and picture
* choose\_from\_radio
  + Given an array of values, creates a group of radio buttons to be part of a form
* **page\_wiki\_map**

Attributes:

* **view** is an int that is the wiki view option

Methods:

* print\_header
  + calls parent print\_header and prints the page title
* print\_content
  + prints out the appropriate menu\_map from the renderer and calls the appropriate print method for the view
* set\_view
  + sets **view**
* set\_url
  + sets the page’s url
* create\_navbar
  + sets PAGE->navbar
* print\_contributions\_content
  + prints out everyone who has contributed and the page they contributed to
* print\_navigation\_content
  + displays in a table the pages that link to this page and the pages this page links to
* print\_index\_content
  + builds a tree using wiki\_build\_tree with the starting node being the current page and then displays the tree
* print\_page\_list\_content
  + gets all the pages in the wiki and displays them in alphabetical order
* print\_orphaned\_content
  + prints a list of all the orphaned pages
* print\_updated\_content
  + gets updated pages using wiki\_get\_updated\_pages\_by\_subwiki and displays them with the picture of the user who edited them
* render\_navigation\_node
  + ceates the html for a node in the index tree
* **page\_wiki\_restoreversion**

Attributes:

* **version** is the version of the page to be restored

Methods:

* print\_header
  + calls parent print\_header and prints the page title
* print\_content
  + checks manage wiki capability and calls print\_restoreversion
* set\_url
  + sets the page’s url
* set\_versionid
  + sets **version**
* create\_navbar
  + sets PAGE->navbar
* setup\_tabs
  + sets the active tab to history
* print\_restoreversion
  + Prints the restore version content
* **page\_wiki\_deletecomment**

Attributes:

* **commentid** the id of the comment to be deleted

Methods:

* print\_header
  + calls parent print\_header and prints the page title
* print\_content
  + calls printconfirmdelete
* set\_url
  + sets the page’s url
* set\_action
  + sets the page’s action commented and content
* create\_navbar
  + setsPAGE->navbar
* setup\_tabs
  + sets the active tab as comments
* printconfirmdelete
  + Prints the comment deletion confirmation form
* **page\_wiki\_save**

Attributes:

* **newcontent** the content from the editor

Methods:

* print\_content
  + calls print\_save
* set\_newcontent
  + sets the new content
* print\_save
  + gets the data from the mod\_wiki\_edit\_form and saves it to the database using either wiki\_save\_section or wiki\_save\_page then redirects to view.php
* **page\_wiki\_viewversion**

Attributes:

* **version** the version row from the database of version to be viewed

Methods:

* print\_header
  + calls parent print\_header and prints the page title
* print\_content
  + checks viewpage wiki capability and calls print\_version\_view
* set\_url
  + sets the page’s url
* set\_versionid
  + sets **version**
* create\_navbar
  + sets PAGE->navbar
* setup\_tabs
  + sets the history tab active
* print\_version\_view
  + prints out the contents of the version
* **page\_wiki\_confirmrestore**

Attributes:

* **version** the version row from the database of version to be restored

Methods:

* set\_url
  + sets the page’s url
* print\_content
  + calls wiki\_restore\_page and redirects to view.php
* set\_versionid
  + sets **version**
* **page\_wiki\_prettyview**

Methods:

* print\_header
  + prints the page header and adds a heading for the title of the page
* print\_content
  + calls print\_pretty\_view
* set\_url
  + sets the page’s url
* print\_pretty\_view
  + gets the version of the page parses the content and echo’
* **page\_wiki\_handlecomments**

Attributes:

* **action** the users action add/edit/delete
* **content** the text for the comment
* **comentid** the id of the comment in the database
* **format** is the wiki page’s format

Methods:

* print\_header
  + calls set\_url
* print\_content
  + if action is add or edit calls add\_comment
  + else calls delete\_comment and redirects to comments.php
* set\_url
  + sets the page’s url
* set\_action
  + sets the page’s action commented format and content
* add\_comment
  + adds a coment by calling wiki\_add\_comment then redirects to comments.php
* delete\_comment
  + deletes a comment by calling wiki\_delete\_comment
* **page\_wiki\_lock**

Methods:

* print\_header
  + calls set\_url
* set\_url
  + set’s the page’s url
* print\_content
  + sets lock by calling wiki\_set\_lock
* **page\_wiki\_overridelocks**

Methods:

* print\_header
  + calls set\_url
* set\_url
  + set’s the page’s url
* print\_content
  + deletes locks by calling wiki\_set\_locks and redirects to edit.php
* **page\_wiki\_admin**

Attributes:

* **view** int for which viewmode the user is using
* **listorphan** Boolean default false used to show orphaned pages

Methods:

* \_\_construct
  + Calls parent \_\_construct and calls init for M.mod\_wiki.deleteversion
* print\_header
  + calls parent print\_header and prints the page title
* print\_content
  + displays an admin menu then calls print\_delete\_content or print\_delete\_content depending on the view
* set\_view
  + sets **view** and **listorphan**
* set\_url
  + sets the page’s url
* create\_navbar
  + sets PAGE->navbar
* print\_delete\_content
  + Shows wiki page delete options
* add\_page\_delete\_options
  + helper function for print\_delete\_content. It will add data to the table.
* print\_delete\_version
  + Prints lists of versions which can be deleted
* choose\_from\_radio

Given an array of values, creates a group of radio buttons to be part of a form helper function for print\_delete\_version