

MediaWiki Developer's Guide (MDG) v1.92

2011-10-18 mwbook #01

How to become a smart developer

Contents

Articles

Coding conventions	1
JavaScript performance	21
jQuery	28
Pre-commit checklist	30
How to debug	31
Requests for comment page: discuss your new idea before committing	35

Code review **37**

Code review	37
Code review guide	38
Code review tags	45

Security **47**

Security	47
Security for developers	48
Security (Manual)	52
Cross-site scripting	58
Cross-site request forgery	60
Register globals	62
SQL injection	65
Database access	66
Securing database passwords	71

Extensions, SpecialPages, hooks & Co. **75**

Extensions	75
Developing extensions	77
Parser functions	84
Special pages, OutputPage, WebRequest, Database, User, Title.php, FAQ	88
Title.php	99
\$wgLegalTitleChars	101
Extending wiki markup	102
Magic words (Manual)	104
Magic words (Help)	107
Hooks	119

Extension hook registry	143
Resource Loader	146
ResourceLoader	146
ResourceLoader/Requirements	147
ResourceLoader/Migration guide for extension developers	150
ResourceLoader/JavaScript Deprecations	153
Testing	169
Testing portal	169
Writing testable PHP code	171
PHP unit testing	172
JavaScript unit testing	176
Parser tests	179
Style guide	180
Style guide	180
Color selection and accessibility	180
Style guide/Forms	182
Error handling and validation	189
Specialized form elements	192
Assets and gallery	195
Localisation - Internationalisation (i18n)	197
Translatewiki.net	197
Localisation	198
Newcomers	212
How to become a MediaWiki hacker	212
Commit access	216
Development policy	223
USERINFO file	225
Subversion	227
References	
Article Sources and Contributors	236
Image Sources, Licenses and Contributors	238
Article Licenses	

Coding conventions

shortcut **CC**

This page describes the **coding conventions** used within the MediaWiki codebase and extensions which are intended for use on Wikimedia websites, including appropriate naming conventions. If you would like a short checklist to help you review your commits, try using the pre-commit checklist.

To help developers fix code with an inadequately spacey style, a tool called `stylize.php`^[1] has been created, which uses PHP's tokenizer extension to add spaces at the relevant places. It's recommended to run this over new files committed to SVN, but not so much for fixing current files.

All languages

Some general guidelines apply to all MediaWiki code, whatever language it is written in.

File formatting

Tab size

Lines should be indented with a single tab character per indenting level. You should make no assumptions about the number of spaces per tab. Most MediaWiki developers find 4 spaces per tab to be best for readability, but many systems are configured to use 8 spaces per tab and some developers might use 2 spaces per tab.

Newlines

All text files should be checked in to Subversion with `svn:eol-style` set to "native". This is necessary to prevent corruption by certain Windows-based text editors.

You can ask subversion to automatically set this property on a given list of file type. This is done with subversion auto-props and will save you the hassle to manually set the property when adding new files.

You might want to read the English Wikipedia article about newline.

Encoding

All text files **must** be encoded with UTF-8 without byte order marks.

Do not use Microsoft Notepad to edit files. Notepad always inserts BOM. BOM will stop PHP files from working since it is a special character inserted at the very top of the file and will be outputted by the web browser to the client.

In short, make sure your editor supports UTF-8 without BOM.

Indenting and alignment

General style

MediaWiki's indenting style is similar to the so-called "One True Brace Style". Braces are placed on the same line as the start of the function, conditional, loop, etc.

```
function wfTimestampOrNull( $outputtype = TS_UNIX, $ts = null ) {
    if ( is_null( $ts ) ) {
        return null;
    } else {
        return wfTimestamp( $outputtype, $ts );
    }
}
```

```
}
```

Multi-line statements are written with the second and subsequent lines being indented by one extra level:

```
return strtolower( $val ) == 'on'
    || strtolower( $val ) == 'true'
    || strtolower( $val ) == 'yes'
    || preg_match( "/^\s*[+-]?0*[1-9]/", $val );
```

Use indenting and line breaks to clarify the logical structure of your code. Expressions which nest multiple levels of parentheses or similar structures may begin a new indenting level with each nesting level:

```
$wgAutopromote = array(
    'autoconfirmed' => array( '&',
        array( APCOND_EDITCOUNT, &$wgAutoConfirmCount ),
        array( APCOND_AGE, &$wgAutoConfirmAge ),
    ),
);
```

There are some exceptions, such as switch statements, where the indentation of the cases are optional, so both of the below are fine.

```
switch ( $mimetype ) {
case 'text/xml':
case 'application/xhtml+xml':
case 'application/xml':
    return true;
default:
    return false;
}
```

```
switch ( $mimetype ) {
    case 'text/xml':
    case 'application/xhtml+xml':
    case 'application/xml':
        return true;
    default:
        return false;
}
```

Vertical alignment

Avoid vertical alignment. It tends to create diffs which are hard to interpret, since the width allowed for the left column constantly has to be increased as more items are added.

Note: Most diff tools provide options to ignore whitespace. For Subversion, the `-X -w` flag makes the `svn diff` command ignore all whitespace.

When needed, create mid-line vertical alignment with spaces. For instance this:

```
$namespaceNames = array(
    NS_MEDIA          => 'Media',
    NS_SPECIAL         => 'Special',
    NS_MAIN           => '',
```

Is achieved as follows with spaces rendered as dots:

```
$namespaceNames .= array(
    → NS_MEDIA.....=> 'Media',
    → NS_SPECIAL.....=> 'Special',
    → NS_MAIN.....=> '',
```

Line continuation

Lines should be broken at between 80 and 100 columns. There are some rare exceptions to this. Functions which take lots of parameters are not exceptions.

The operator separating the two lines may be placed on either the following line or the preceding line. An operator placed on the following line is more visible and so is more often used when the author wants to draw attention to it:

```
return strtolower( $val ) == 'on'
    || strtolower( $val ) == 'true'
    || strtolower( $val ) == 'yes'
    || preg_match( "/^\s*[+-]?0*[1-9]/", $val );
```

An operator placed on the preceding line is less visible, and is used for more common types of continuation such as concatenation and comma:

```
$wgOut->addHTML(
    Xml::fieldset( wfMsg( 'importinterwiki' ) ) .
    Xml::openElement( 'form', array( 'method' => 'post', 'action' =>
$action, 'id' => 'mw-import-interwiki-form' ) ) .
    wfMsgExt( 'import-interwiki-text', array( 'parse' ) ) .
    Xml::hidden( 'action', 'submit' ) .
    Xml::hidden( 'source', 'interwiki' ) .
    Xml::hidden( 'editToken', $wgUser->editToken() ) .
);
```

When continuing "if" statements, a switch to Allman-style braces makes the separation between the condition and the body clear:

```
if ( ( $this->mNamespace != NS_SPECIAL && strlen( $dbkey ) > 255 )
    || strlen( $dbkey ) > 512 )
{
    return false;
```

```
}
```

Opinions differ on the amount of indentation that should be used for the conditional part. Using an amount of indentation different to that used by the body makes it more clear that the conditional part is not the body, but this is not universally observed.

Continuation of conditionals and very long expressions tend to be ugly whichever way you do them. So it's sometimes best to break them up by means of temporary variables.

Spaces

MediaWiki favours a heavily-spaced style for optimum readability.

Put spaces on either side of binary operators, for example:

```
$a = $b + $c;
```

not like this:

```
$a=$b+$c;
```

Put spaces next to parentheses on the inside, except where the parentheses are empty. Do not put a space following a function name.

```
$a = getFoo( $b );  
$c = getBar();
```

Opinions differ as to whether control structures `if` , `while` , `for` and `foreach` should be followed by a space; the following two styles are acceptable:

```
// Spacey  
if ( isFoo() ) {  
    $a = 'foo';  
}  
  
// Not so spacey  
if( isFoo() ) {  
    $a = 'foo';  
}
```

Single-line comments should have a space between the `#` or `//` and the comment text.

To help developers fix code with an inadequately spacey style, a tool called `stylize.php` ^[1] has been created, which uses PHP's tokenizer extension to add spaces at the relevant places.

Braceless control structures

Do not use single-line `if` statements:

```
if ( $done ) return;
if ( $done ) { return; }
```

They reduce the readability of the code by moving important statements away from the left margin, where the reader is looking for them. Remember that making code shorter doesn't make it simpler. The goal of coding style is to communicate effectively with humans, not to fit computer-readable text into a small space.

Most MediaWiki developers favour fully-braced control structures:

```
if ( $done ) {
    return;
}
```

This avoids a common logic error, which is especially prevalent when the developer is using a text editor which does not have a "smart indenting" feature. The error occurs when a single-line block is later extended to two lines:

```
if ( $done )
    return;
```

Later changed to:

```
if ( $done )
    $this->cleanup();
    return;
```

This has the potential to create subtle bugs.

emacs style

In emacs, using `php-mode.el` from `nXHTML mode`^[2], you can set up a MediaWiki minor mode in your `.emacs` file:

```
(defconst mw-style
  '( (indent-tabs-mode . t)
    (tab-width . 4)
    (c-basic-offset . 4)
    (c-offsets-alist . ((case-label . +)
                        (arglist-cont-nonempty . +)
                        (arglist-close . 0)
                        (cpp-macro . (lambda (x) (cdr x)))
                        (comment-intro . 0)))
    (c-hanging-braces-alist
      (defun-open after)
      (block-open after)
      (defun-close)))

(c-add-style "MediaWiki" mw-style)
```

```

(define-minor-mode mw-mode
  "tweak style for mediawiki"
  nil " MW" nil
  (delete-trailing-whitespace)
  (tabify (point-min) (point-max))
  (c-subword-mode 1))

;; Add other sniffers as needed
(defun mah/sniff-php-style (filename)
  "Given a filename, provide a cons cell of
  (style-name . function)
where style-name is the style to use and function
sets the minor-mode"
  (cond ((string-match "\\(mw[/]*\\|mediawiki\\)/"
                      filename)
         (cons "MediaWiki" 'mah/mw-mode))
        (t
         (cons "cc-mode" (lambda (n) t))))))

(add-hook 'php-mode-hook (lambda () (let ((ans (when (buffer-file-name)
                                                    (mah/sniff-php-style
 (buffer-file-name)))))
                                (c-set-style (car ans))
                                (funcall (cdr ans) 1))))))

```

The above `mah/sniff-php-style` function will check your path when `php-mode` is invoked to see if it contains “mw” or “mediawiki” and set the buffer to use the `mw-mode` minor mode for editing MediaWiki source. You will know that the buffer is using `mw-mode` because you'll see something like “PHP MW” or “PHP/lw MW” in the mode line.

File naming

Files which contain server-side code should be named in *UpperCamelCase*. This is also our naming convention for extensions.^[3] Name the file after the most important class it contains; most files will contain only one class, or a base class and a number of descendants. For instance, `Title.php` contains only the `Title` class; `HTMLForm.php` contains the base class `HTMLForm`, but also the related class `HTMLFormField` and its descendants.

Name 'access point' files, such as SQL, and PHP entry points such as `index.php` and `foobar.sql`, in *lowercase*. Maintenance scripts are generally in *lowerCamelCase*, although this varies somewhat. Files intended for the site administrator, such as readmes, licenses and changelogs, are usually in *UPPERCASE*.

Never include spaces in file names or directories, and never use non-ASCII characters. For lowercase titles, hyphens are preferred to underscores.

For JS and CSS files loaded via `ResourceLoader`, file naming should match module naming. For example:

- module `mediawiki.foo` has files `resources/mediawiki.foo/mediawiki.foo.js` and `resources/mediawiki.foo/mediawiki.foo.css`
- module `mediawiki.Title` has file `resources/mediawiki/mediawiki.Title.js`

JS and CSS files for extensions usually use a name like `ext.myExtension`, for instance:

- `extensions/FooBar/resources/ext.fooBar/ext.fooBar.init.js`

This keeps it easy to find things, even if you divide up a module into smaller files for editing convenience and then bundle them together into a single module.

Groups of modules should have their files also grouped in directories. For example, there are several modules related to "jquery". All those modules start with "jquery." and are stored in the `resources/jquery` directory.

PHP code

Code structure

Assignment expressions

Using assignment as an expression is surprising to the reader and looks like an error. Do not write code like this:

```
if ( $a = foo() ) {  
    bar();  
}
```

Space is cheap, and you're a fast typist, so instead use:

```
$a = foo();  
if ( $a ) {  
    bar();  
}
```

Using assignment in a `while()` clause used to be legitimate, for iteration:

```
$res = $dbr->query( 'SELECT * FROM some_table' );  
while ( $row = $dbr->fetchObject( $res ) ) {  
    showRow( $row );  
}
```

This is unnecessary in new code; instead use:

```
$res = $dbr->query( 'SELECT * FROM some_table' );  
foreach ( $res as $row ) {  
    showRow( $row );  
}
```

Ternary operator

The ternary operator ^[4] can be used profitably if the expressions are very short and obvious:

```
$wiki = isset( $this->mParams['wiki'] ) ? $this->mParams['wiki'] : false;
```

But if you're considering a multi-line expression with a ternary operator, please consider using an `if()` block instead. Remember, disk space is cheap, code readability is everything, "if" is English and `?:` is not.

5.3 shorthand

Since we still support PHP 5.2.x, use of the shorthand ternary operator introduced in PHP 5.3 is not allowed.

String literals

For simple string literals, single quotes are slightly faster for PHP to parse than double quotes. Perhaps more importantly, they are easier to type, since you don't have to press shift. For these reasons, single quotes are preferred in cases where they are equivalent to double quotes.

However, do not be afraid of using PHP's double-quoted string interpolation feature: `$elementId = "myextension-$index";` This has slightly better performance characteristics than the equivalent using the concatenation (dot) operator, and it looks nicer too.

Heredoc-style strings are sometimes useful:

```
$s = <<<EOT
<div class="mw-some-class">
$boxContents
</div>
EOT;
```

Some authors like to use `END` as the ending token, which is also the name of a PHP function. This leads to IRC conversations like the following:

```
<Simetrical>      vim also has ridiculously good syntax highlighting.
<TimStarling>     it breaks when you write <<<END in PHP
<Simetrical>      TimStarling, but if you write <<<HTML it syntax-highlights as HTML!
<TimStarling>     I have to keep changing it to ENDS so it looks like a string again
<brion-codereview> fix the bug in vim then!
<TimStarling>     brion-codereview: have you ever edited a vim syntax script file?
<brion-codereview> hehehe
<TimStarling>     http://tstarling.com/stuff/php.vim
<TimStarling>     that's half of it...
<TimStarling>     here's the other half: http://tstarling.com/stuff/php-syntax.vim
<TimStarling>     1300 lines of sparsely-commented code in a vim-specific language
<TimStarling>     which turns out to depend for its operation on all kinds of subtle inter-pass effects
<werdnum>         TimStarling: it looks like some franken-basic language.
```

Functions and parameters

Avoid passing huge numbers of parameters to functions or constructors:

```
//Constructor for Block.php as of 1.17. *DON'T* do this!
function __construct( $address = '', $user = 0, $by = 0, $reason = '',
    $timestamp = 0, $auto = 0, $expiry = '', $anonOnly = 0,
    $createAccount = 0, $enableAutoblock = 0,
    $hideName = 0, $blockEmail = 0, $allowUsertalk = 0 )
{
    ...
}
```

It quickly becomes impossible to remember the order of parameters, and you will inevitably end up having to hardcode all the defaults in callers just to customise a parameter at the end of the list. If you are tempted to code a function like this, consider passing an associative array of named parameters instead.

In general, using boolean parameters is discouraged in functions. In `$object->getSomething($input, true, true, false)`, without looking up the documentation for `MyClass::getSomething()`, it is impossible to know what those parameters are meant to indicate. Much better is to either use class constants, and make a generic flag parameter:

```
$myResult = MyClass::getSomething( $input, MyClass::FROM_DB &
    MyClass::PUBLIC_ONLY );
```

Or to make your function accept an array of named parameters:

```
$myResult = MyClass::getSomething( $input, array( 'fromDB',
    'publicOnly' ) );
```

Try not to repurpose variables ^[5] over the course of a function, and avoid modifying the parameters passed to a function (unless they're passed by reference and that's the whole point of the function, obviously).

C borrowings

The PHP language was designed by people who love C and wanted to bring souvenirs from that language into PHP. But PHP has some important differences from C.

In C, constants are implemented as preprocessor macros and are fast. In PHP, they are implemented by doing a runtime hashtable lookup for the constant name, and are slower than just using a string literal. In most places where you would use an enum or enum-like set of macros in C, you can use string literals in PHP.

PHP has three special literals: `true`, `false` and `null`. Homesick C developers write `null` as `NULL` because they want to believe that it is a macro defined as `((void*) 0)`. This is not necessary.

Use `elseif` not `else if`. They have subtly different meanings ^[6]:

```
// This:
if( $foo == 'bar' ) {
    echo 'Hello world';
} else if( $foo == 'Bar' ) {
    echo 'Hello world';
} else if( $baz == $foo ) {
    echo 'Hello baz';
}
```

```

} else {
    echo 'Eh?';
}

// Is actually equivalent to:
if( $foo == 'bar' ) {
    echo 'Hello world';
} else {
    if( $foo == 'Bar' ) {
        echo 'Hello world';
    } else {
        if( $baz == $foo ) {
            echo 'Hello baz';
        } else {
            echo 'Eh?';
        }
    }
}
}

```

And the latter has poorer performance.

PHP pitfalls

- Understand and read the documentation for `isset()` ^[7] and `empty()` ^[8]. Use them only when appropriate.
 - `empty()` is inverted conversion to boolean with error suppression. Only use it when you really want to suppress errors. Otherwise just use `!`. Do not use it to test if an array is empty, unless you simultaneously want to check if the variable is unset.
 - **Do not use `isset()` to test for `null`**. Using `isset()` in this situation could introduce errors by hiding mis-spelled variable names. Instead, use `$var === null`
- Study the rules for conversion to boolean ^[9]. Be careful when converting strings to boolean.
- Be careful with double-equals comparison operators ^[10]. Triple-equals is often more intuitive.
 - `'foo' == 0` is true
 - `'000' == '0'` is true
 - `'000' === '0'` is false
- Array plus ^[11] does not renumber the keys of numerically-indexed arrays, so `array('a') + array('b')` `=== array('a')`. If you want keys to be renumbered, use `array_merge()` ^[12]: `array_merge(array('a'), array('b')) == array('a', 'b')`
- Make sure you have `error_reporting` ^[13] set to `E_ALL` for PHP 5. This will notify you of undefined variables and other subtle gotchas that stock PHP will ignore. See also Manual:How to debug.
- When working in a pure PHP environment, remove any trailing `?>` tags. These tags often cause issues with trailing white-space and "headers already sent" error messages (cf. bugzilla:17642 and <http://news.php.net/php.general/280796>).
- Do not use the 'goto' syntax introduced in 5.3. PHP may have introduced the feature, but that does not mean we should use it.

Integration

There are a few pieces of code in the MediaWiki codebase which are intended to be standalone and easily portable to other applications; examples include the UTF normalisation in `/includes/normal` and the libraries in `/includes/libs`. Apart from these, code should be integrated into the rest of the MediaWiki environment, and should allow other areas of the codebase to integrate with it in return.

Global objects

Main page: `Manual:RequestContext.php`

Do not access the PHP superglobals `$_GET` , `$_POST` , etc, directly; use ^[14]`$request->get*('param')` instead; there are various functions depending on what type of value you want. You can get a `WebRequest` from the nearest `RequestContext` , or if absolutely necessary `$wgRequest` . Equally, do not access `$_SERVER` directly; use `$request->getIP()` if you want to get the IP address of the current user.

Classes

Encapsulate your code in an object-oriented class, or add functionality to existing classes; do not add new global functions or variables. Try to be mindful of the distinction between 'backend' classes, which represent entities in the database (eg `User`, `Block`, `Revision`, etc), and 'frontend' classes, which represent pages or interfaces visible to the user (`SpecialPage`, `Article`, `ChangesList`, etc. Even if your code is not obviously object-oriented, you can put it in a static class (eg `IP` or `Html`) .

As a holdover from PHP 4's lack of private class members and methods, older code will be marked with comments such as `/** @private */` to indicate the intention; respect this as if it were enforced by the interpreter.

Mark new code with proper visibility modifiers ^[15], including `public` if appropriate, but **do not** add visibility to existing code without first checking, testing and refactoring as required. It's generally a good idea to avoid visibility changes unless you're making changes to the function which would break old uses of it anyway.

Error handling

Don't suppress errors with PHP's `@` operator ^[16], for any reason ever ^[17]. It's broken when `E_STRICT` is enabled and it causes an unlogged, unexplained error if there is a fatal, which is hard to support. Use `wfSuppressWarnings()` ^[18] and `wfRestoreWarnings()` ^[19] instead. The `checkSyntax.php` maintenance script can check for this error for you.

When your code encounters a sudden error, you should throw a `MWException` (or an appropriate subclass) rather than using PHP's `trigger_error`. The exception handler will display this as nicely as possible to the end user and wiki administrator, and also provides a stack trace to developers.

PHP Naming

Use `lowerCamelCase` when naming functions or variables. For example:

```
private function doSomething( $userPrefs, $editSummary )
```

Use `UpperCamelCase` when naming classes: `class ImportantClass` . Use uppercase with underscores for global and class constants: `DB_MASTER` , `Revision::REV_DELETED_TEXT` . Other variables are usually lowercase or `lowerCamelCase`; avoid using underscores in variable names.

There are also some prefixes used in different places:

Functions

- `wf` (wiki functions) – top-level functions, e.g.

```
function wfFuncname() { ... }
```

Verb phrases are preferred: use `getReturnText()` instead of `returnText()` .

Variables

- `$wg` – global variables, e.g. **`$wgVersion`**, **`$wgTitle`**. Always use this for new globals, so that it's easy to spot missing "global `$wgFoo` " declarations. In extensions, the extension name should be used as a namespace delimiter. For example, `$wgAbuseFilterConditionLimit` , **not** `$wgConditionLimit` .

It is common to work with an instance of the `Database` class; we have a naming convention for these which helps keep track of the nature of the server to which we are connected. This is of particular importance in replicated environments, such as Wikimedia and other large wikis; in development environments there is usually no difference between the two types, which can conceal subtle errors.

- `$dbw` – a `Database` object for writing (a master connection)
- `$dbr` – a `Database` object for non-concurrency-sensitive reading (this may be a read-only slave, slightly behind master state, so don't ever try to write to the database with it, or get an "authoritative" answer to important queries like permissions and block status)

The following may be seen in old code but are discouraged in new code:

- `$ws` – Session variables, e.g. `$_SESSION['wsSessionName']`
- `$wc` – Cookie variables, e.g. `$_COOKIE['wcCookieName']`
- `$wp` – Post variables (submitted via form fields), e.g. `$wgRequest->getText('wpLoginName')`
- `$m` – object member variables: `$this->mPage` . This is **discouraged in new code**, but try to stay consistent within a class.

Documentation

Inline

The Doxygen documentation style is used (it is very similar to PHPDoc for the subset that we use). A code documentation example: giving a description of a function or method, the parameters it takes (using `@param`), and what the function returns (using `@return`), or the `@ingroup` or `@author` tags.

Use `@` rather than `\` as the escape character (i.e. use `@param` rather than `\param`) – both styles work in Doxygen, but only `@param` style works with PHPDoc. Use `/**` to begin the comments, instead of the Qt-style formatting `/*!`.

General format for parameters is such: `@param [type] $varname [description]`. Multiple types can be listed by separating with a pipe character.

```
/**
 * Get the text that needs to be saved in order to undo all revisions
 * between $undo and $undoafter. Revisions must belong to the same
page,
 * must exist and must not be deleted
 * @param $undo Revision
 * @param $undoafter Revision Must be an earlier revision than $undo
 * @return mixed string on success, false on failure
```



```

*/
public function getUndoText( Revision $undo, Revision $undoafter = null
) {
    $undo_text = $undo->getText();
    $undoafter_text = $undoafter->getText();
    $cur_text = $this->getContent();
    if ( $cur_text == $undo_text ) {
        # No use doing a merge if it's just a straight revert.
        return $undoafter_text;
    }
    $undone_text = '';
    if ( !wfMerge( $undo_text, $undoafter_text, $cur_text,
$undone_text ) ) {
        return false;
    }
    return $undone_text;
}

```

PHPDoc was used at the very beginning but got replaced with Doxygen for performance reason. We should probably drop PHPDoc compatibility.

There is a bug in Doxygen that affect MediaWiki's documentation:

- Bug 626105 - @var in php is not documented ^[20]
@var should be usable to document class members' type, but doesn't work for now.

/docs folder

Some elements of MediaWiki are documented in the `/docs` folder ^[21]. For instance, if you add a new hook, you should update docs/hooks.txt with the name of the hook, a description of what the hook does, and the parameters used by the hook.

Release notes

All significant changes ^[22] to the core software which might affect wiki users ^[22], server administrators, or extension authors ^[23], must be documented in the `RELEASE-NOTES` file. This file is refreshed on every release (with the past content going into `HISTORY`) and is generally divided into three sections:

- **Configuration changes** is the place to put changes to accepted default behaviour, backwards-incompatible changes, or other things which need a server administrator to look at and *decide* "is this change right for my wiki?". Try to include a *brief* explanation of how the previous functionality can be recovered if desired.
- **Bug fixes** is the place to note changes which fix behaviour which is accepted to be problematic or undesirable. These will often be issues reported in bugzilla, but needn't necessarily.
- **New features** is, unsurprisingly, to note the addition of new functionality.

In all cases, if your change is in response to an issue reported in bugzilla, include the bug reference at the start of the entry. New entries are added in chronological order at the end of the section ^[24].

Messages

When creating a new message, use hyphens (-) where possible ^[25] instead of CamelCase or rails_underscored_variables. So for example, some-new-message is a good name, while someNewMessage and some_new_message are not.

If the message is going to be used as a label which can have a colon (:) after it, don't hardcode the colon; instead, put the colon inside the message text. Some languages (such as French which require a space before) need to handle colons in a different way, which is impossible if the colon is hardcoded.

Try to use message keys "whole" in code, rather than building them on the fly; as this makes it easier to search for them in the codebase. For instance,

```
return wfMsg( 'templatesused-' . ( $section ? 'section' : 'page' ) ) );
```

is bad because a search for templatesused-section will not find this use of the message key. Instead do:

```
$key = $section ? 'templatesused-section' : 'templatesused-page';
return wfMsg( $key );
```

MySQL

Use *UPPERCASE* for MySQL keywords, and *lowercase* for things like types. Do not specify the lengths of numeric types, but do for *varchar()* and *varbinary()* types. Use *varchar(14)* for all timestamps, and parse them to the standard format using *\$dbw->timestamp(\$ts)*; do not use the *timestamp* field type.

Make sure to include the */*_*/* comment immediately before any table name; this will be replaced with the wiki's database prefix if necessary, and omitting it will cause breakage. Similarly, include the */*\$wgDBTableOptions*/* comment after any table declaration, and */*i*/* immediately before any index names.

Create indices as separate statements, do not include them in the table creation query; the separate syntax is clearer and makes it easier to see the difference between unique and non-unique indices. Don't create indices with **ALTER TABLE ... ADD INDEX ...**, always use **CREATE INDEX ... ON ...** instead.

```
--
-- Track page-to-page hyperlinks within the wiki.
--
CREATE TABLE /*_*/pagelinks (
    -- Key to the page_id of the page containing the link.
    pl_from int unsigned NOT NULL default 0,

    -- Key to page_namespace/page_title of the target page.
    -- The target page may or may not exist, and due to renames
    -- and deletions may refer to different page records as time
    -- goes by.
    pl_namespace int NOT NULL default 0,
    pl_title varchar(255) binary NOT NULL default ''
) /*$wgDBTableOptions*/;

CREATE UNIQUE INDEX /*i*/pl_from ON /*_*/pagelinks
(pl_from,pl_namespace,pl_title);
```

```
CREATE UNIQUE INDEX /*i*/pl_namespace ON /*_*/pagelinks
(pl_namespace,pl_title,pl_from);
```

Table naming

- Table names should be singular nouns: user, page, revision, etc. There are some historical exceptions: pagelinks, categorylinks...
- Column names are given a prefix derived from the table name: the name itself if it's short, or an abbreviation:
 - page → page_id, page_namespace, page_title...
 - categorylinks → cl_from, cl_namespace...

ResourceLoader

Module naming

Module names should match the main definition of the scripts they load. For example a module defining the the `mw.util` object is named "mediawiki.util" and the module for the `mw.Title` object constructor is named "mediawiki.Title".

See also:

- #File naming
- #JavaScript naming

JavaScript code

See also JavaScript performance

In most cases conventions for PHP can perfectly apply to JavaScript code. Following are JavaScript specific conventions.

If making use of a scope (ie. local variables), always wrap your code in a closure. This avoids leaking variables to, or being given local variables from, another module. Use one of the following constructions ("Immediately-Invoked Function Expression"^[26]):

```
( function() {
    // Simple closure.
    // Code here will be invoked immediately
} )();

( function( $ ) {
    // Simple closure
    // Code here will be invoked immediately with jQuery as first
    argument ($)
} )( jQuery );

jQuery( document ).ready( function( $ ) {
    // Function bound to document ready queue
    // This function will not be invoked immediately,
    // instead it is called by jQuery when the document is ready

    // jQuery( fn ), is a shortcut to jQuery( document ).ready( fn )
} );
```

jQuery

See also jQuery

jQuery is now provided as a standard library in MediaWiki 1.17; familiarize yourself with its API ^[27] and don't be afraid to use jQuery for DOM manipulation, AJAX requests, and utility functions. These are usually much easier to work with than classic low-level DOM and DHTML interfaces, and much more consistent across browsers.

To avoid confusion with raw elements or other variables, we prefix variables storing instances of jQuery with a dollar symbol. This makes it easy to recognize and manipulate them even with great distance between point X and the point of definition. A common problem is when referring to them in an `if` statement. `document.getElementById` returns null if no element matched the ID, therefore (since null casts to boolean false) it can be used as-is in an if statement. jQuery objects on the other hand (as any object in JavaScript) cast to boolean true no matter what.

JavaScript naming

Naming of functions, variables, constructor functions etc. is pretty much the same as we do in PHP.

Use lowerCamelCase when naming functions or variables. For example:

```
function doSomething( a, b )
```

```
var fooBar = 'Hello';
```

```
obj.getStuff: function( a, b ) { ...
```

Use UpperCamelCase when naming object constructors.

jQuery pitfalls

- As recorded under jqbug 8229 ^[28], the attribute selector values should always be quoted.

```
$ ( '#pagehistory li input[name=diff]' ); // Bad
$ ( '#pagehistory li input[name="diff"]' ); // Good!

$ ( '#foo' ).find( 'input[name=test[]]' ); // Very bad
$ ( '#foo' ).find( 'input[name="test[]"]' ); // Good!
```

- `jQuery.fn.size()` is the same number of characters as `.length`, using `.length` saves function call overhead (view source of `.size()` ^[29]). Remember to always use strict equals when comparing the `.length` attribute as 0 and 1 can cast to booleans.

JavaScript pitfalls

- Do not assume the existence of any global variables other than `jQuery` and `mw`. **\$ should always be aliased locally** (see below).
- Array-like access on a string (`var str = 'foobar'; str[3]`) doesn't work in older versions of IE such as IE6. Use the *String* method `charAt` instead (`var str = 'foobar'; str.charAt(3)`).
- Be sure to check the MDN Docs ^[30] before using a prototype. The `filter` ^[31] prototype for Arrays, for example, isn't supported in any version of Internet Explorer before 9. See also `Compatibility#Browser`.
- Always define variables with a clear statement on whether they are local or global (`var foo =` or `window.foo =`). Especially since `ResourceLoader` combines several scripts into one file, wrapped in a self-executing anonymous function. Never make an assumption or imply its scope. Undeclared variables are

assumed to be global - the opposite of PHP.

- When throwing exceptions, use `throw new Error('message')` rather than `throw 'message'`. You can throw a string directly in JavaScript, but some debuggers won't pick up a stack trace or the location of the error without using the `Error` constructor.
- Checking for Arrays or Functions should be done through `$.isArray()` ^[32] and `$.isFunction` ^[33] at all times to cover for edge cases and cross-browser differences.
- Be careful to preserve compatibility with left-to-right **and right-to-left languages** (ie. `float: right` or `text-align: left`), especially when styling text containers. This is another reason why such declarations should be in css files, so that they are automatically flipped by ResourceLoader in RTL-languages.
- When using an object literal, don't include a trailing comma. `var obj = { a: 1, b: 2, };` will fail in some versions of IE while `var obj = { a: 1, b: 2 };` will work universally.
- Similarly, when using an array literal, don't include any extra commas. In older versions of IE, `[1, 2, , 3,]` will be interpreted as `[1, 2, undefined, 3, undefined]` instead of the expected `[1, 2, 3]`.
- Be sure to use `attr()` ^[34] or `prop()` ^[35] the right way. See also <http://javascript.info/tutorial/attributes-and-custom-properties>.

Best practices

- Don't apply styling to lots of elements at once; this has poor performance. Instead use a common parent's class (or add one) and apply CSS in a .css file. Thanks to ResourceLoader, this will all be loaded in the same HTTP request, so there's no performance penalty for having a separate CSS file. Do not set CSS into inline "style" attributes, don't insert "style" elements from JS either.
- Use dot-notation whenever possible to access object members. (`myObj.member` instead of `myObj['member']`)
- Validate with JSHint ^[36] as much as possible. Recommended JSHint settings:
Uncheck everything, except these:
[x] Require curly braces around all blocks;
[x] Don't check line breaks;
[x] Browser
- Use array and object literal notation (`var foo = [];` `var bar = {};`), do not use `new Object()` or `new Array()`.
- `if`, `for`, `while` and the like do not create new contexts, avoid declaring variables inside them. Instead declare them outside and if needed (re)define them inside.
- Don't use multiple `var` statements, combine them when possible and indent with a tab.
- Be careful when validating the type of a variable. For example `Array.isArray` is not natively supported yet in all browsers, for details see jQuery's guideline page: http://docs.jquery.com/JQuery_Core_Style_Guidelines#Type_Checks.

Whitespace

- Like CSS declarations, in an object literal the notation should have no space before the colon and one space after the colon.
- Use operators such as `typeof`^[37] like `typeof foobar`, not like a function.

Commenting and documentation

Document functions and variables (see #Documentation). Make sure though that you use the JavaScript terminology (ie. not `bool` but `Boolean`)

In older versions of MediaWiki, JavaScript code was often very poorly commented to keep it small. Since MediaWiki 1.17 and higher run code through ResourceLoader's minification helpers by default, please feel free to liberally comment your code to aid future maintainers.

Doxygen/jsdoc documentation is not yet built automatically from JavaScript files, but it may be at some point in the future. In the meantime, some advanced editors like NetBeans can make use of jsdoc-style annotations to aid in autocomplete and code navigation.

Here's the format we aim to follow from now on:

```
/**
 * Create an outsources Cool object.
 * @constructor
 *
 * @example
 *     new Cool( 5 );
 *
 * @param foo {Number} Total sources.
 */
function Cool( foo ) {
    // ...
}

/**
 * Some short description of this function.
 *
 * @example
 *     $( '.foo' ).foobar();
 *
 * @context {jQuery}
 * @return {String} HTML-fragment
 */
$.fn.foobar = function() {
    if ( this.val() == 'baz' ) {
        return this.html();
    }
    return this.html( '<baz>Foo foo foo</baz>' ).html();
}
```

CSS

Whitespace

- One selector per line
- Opening braces for the CSS rule on the same line as the (last) selector
- Indent each declaration with a tab
- No space before the colon
- 1 space between the colon and the value
- a semi-colon (;) after each declaration (including the last one)
- Closing braces unindented back to the left
- Annotation for CSSJanus and CSSMin should be on their own line, above the css declaration they're for.
- An empty line between one CSS rule and the next.

```
.selector,
#some-element {
    float: right;
    text-align: left;
}

#look-at-the-left {
    /* @noflip */
    float: left;
    /* @embed */
    background-image: url(images/foobar.png);
}
```

Quotes

In the `background-image` declaration, the `url()` syntax is preferred to be used without quotes. They are not needed. The only case where it could cause problems is when an unescaped closing parenthesis occurs in the given path, but those should be url escaped.

CSS Naming

Class and IDs are named the same way. All css hooks are named lowercase and broken up by dashes. Using the `mw-` prefix avoids conflicts with IDs generated by the wikitext parser to mark section headings

Some examples:

```
/* Site-wide elements */
.mw-body
#mw-panel
#mw-js-message
.mw-headline
.mw-label
.mw-input

/* Special pages */
.mw-spcontent
/* - Special:AllPages */
```

```
.mw-allpages-table-form
.mw-allpages-nav
/* - Special:BlockIp */
#mw-bi-target
```

References

- [1] <http://svn.wikimedia.org/svnroot/mediawiki/trunk/tools/code-utils/stylize.php>
- [2] <http://ourcomments.org/Emacs/nXhtml/doc/nxhtml.html>
- [3] <http://lists.wikimedia.org/pipermail/wikitech-l/2011-August/054839.html>
- [4] <http://php.net/ternary#language.operators.comparison.ternary>
- [5] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-November/027618.html>
- [6] <http://php.net/elseif>
- [7] <http://php.net/isset>
- [8] <http://php.net/empty>
- [9] <http://php.net/boolean.php#language.types.boolean.casting>
- [10] <http://php.net/operators.comparison>
- [11] <http://php.net/operators.array>
- [12] <http://php.net/array-merge>
- [13] http://php.net/error_reporting
- [14] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/030029.html>
- [15] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/029435.html>
- [16] <http://php.net/operators.errorcontrol>
- [17] <http://lists.wikimedia.org/pipermail/wikitech-l/2010-August/048780.html>
- [18] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#a875ea3710563af3ff4f09cd9411e4158
- [19] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#a5c3700e80dde3839ed39b5f2bd674504
- [20] https://bugzilla.gnome.org/show_bug.cgi?id=626105
- [21] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/docs>
- [22] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-August/026207.html>
- [23] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-December/028133.html>
- [24] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-March/030539.html>
- [25] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-December/035583.html>
- [26] <http://benalman.com/news/2010/11/immediately-invoked-function-expression/>
- [27] <http://api.jquery.com/>
- [28] <http://jbug.com/8229>
- [29] <https://github.com/jquery/jquery/blob/1886d7443453feab0b73f4a7c4b15fbd9401c4af/src/core.js#L200>
- [30] <https://developer.mozilla.org/>
- [31] https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/filter
- [32] <http://api.jquery.com/jQuery.isArray/>
- [33] <http://api.jquery.com/jQuery.isFunction/>
- [34] <http://api.jquery.com/attr>
- [35] <http://api.jquery.com/prop>
- [36] <http://jshint.com/>
- [37] <https://developer.mozilla.org/en/JavaScript/Reference/Operators/Special/typeof>

JavaScript performance

shortcut JSPERF

This page lists some common misconceptions and solutions to improve **JavaScript performance** or proper code in general. The intention is to speed up our code base and to make sure code does what it intends. For example `typeof myArray === 'object'` returns true, for starters.

Selector performance (jQuery)

Just like the browser's stylesheet engine that parses native CSS, jQuery's selector engine (*Sizzle*) selects from **right to left**. Let's understand what this means for the following selector which may look fast / specific but in fact is not.

```
jQuery( 'div#bodyContent ul li a' );
```

This particular example may be extreme, but consider the individual tips and tricks.

The above example will initiate the Sizzle engine, starting by getting all Anchor tags on the entire page (there's quite a few! Over 1300 on an average edit page on a Wikimedia wiki, don't forget about hidden links and anchors for javascript actions). After getting all those it will check these one by one to see if their parentNode is a ListItem. If it's not it'll get the parentNode of the parentNode and so on, until it's all the way to the document.body and gives up and get's started on the next anchor tag. The anchor tags in the Sidebar and in the Personal-area will match for a while for ListItem and UnorderedList and eventually they are dropped when it comes to the #bodyContent (since the sidebar and personal-area are not descendants of #bodyContent). After that it'll still continue for the few anchor tags left in the match, eventhough there is only one #bodyContent it'll do one last check in the selector since div was part of the selector as well.

Sizzle has a number of optimization shortcuts for re-rooting queries in certain known forms, and on browsers that support querySelectorAll^[1] in many cases (such as this one) the entire query is handed off to native browser code bypassing the Sizzle engine entirely.

So how to make this more efficient ? The answer is context. Context means to get elements from within the context of one or more elements, rather than from the entire document. In plain javascript this is how that goes:

```
/*
<div id="foo">
  <p>Hello <strong>world</strong>!<br />
  Did you know how <strong>Awesome</strong> <span class="package">MediaWiki</span> is?</p>
</div>
<div id="bar">
  <p><strong>Byebye</strong></p>
</div>
*/
var foo = document.getElementById( 'foo' );
var fooStrong = foo.getElementsByTagName( 'strong' );
// ^ only selects [ <strong>world</strong>, <strong>Awesome</strong> ]
// ... not <strong>Byebye</strong>! because it's not in the context of #foo
```

So the example given is much faster like this due to context find:

```
$( 'div#bodyContent' ).find( 'ul li a' );
```

You can optimize even further (depending on your scenario) like this:

```
$( '#bodyContent' ).find( 'li > a' );
```

- Dropped the `div` selector.
- Removed `ul` from the selector. Unless you expect to have invalid `ListItems` without a list around them or if you specifically need to filter out `OrderedLists` (``), there's no need to specify each element in the tree.
- Added a direct-child expression to speed up the filtering of anchor tags. Ofcourse if your situation may contain `Foo bar baz lorem` then you don't want this. But whenever you can, use the direct child expression.

Generally one could describe writing more efficient selectors as *"Be specific in what you want, and do not baby the selector engine"*. Specify what you really want, don't add in more crap if that's not what you need (ie. stuff that just happens to be the case like the extra "div" in front of the ID and the "ul" in between. Unless, as noted, if you specifically need the "div" or "ul" in which case it's fine to add them in the selector).

Null or undefined

There are more than a dozen ways of comparing or looking at a variable to determine whether it is null or undefined (or either of them). Look at the following for actual performance numbers and decide for yourself what gives the best performance for your scenario:

- <http://jsperf.com/testing-undefined>
- <http://jsperf.com/testing-null>

Argument or object property undefined

The fastest way to check if an (un)passed argument or object property is undefined is by strictly comparing it to the `undefined` literal.

```
function bar( lorem ) {
  if ( lorem === undefined ) {
    // ...
  }
}

if ( myObject.property === undefined ) {
  // ...
}
```

What about `typeof` ?

So why is a `typeof` comparison slow ? The reason is that there are a few things going on. For one, the `typeof` operator returns a string which is a fairly expensive operation. So what really happens is an operation followed by a string comparison. Check the jsperf's above to compare results.

Variable undeclared

So why do people use `typeof`? It's often used to avoid throwing a `ReferenceError` if the variable was undeclared. However in pretty much every scenario the variable isn't undeclared, just *undefined*. The two can be confusing, but once you know it will make your code cleaner, simpler and faster.

Arguments and object properties are never considered undeclared. The only scenario in which referring to a variable throws an exception is when the variable is undeclared (in addition to being undefined). And the only way that can happen is by referring to a variable that has no "var" declaration in the local scope, and isn't a global variable. Which should be rare in most cases, and in good quality code not happen at all. See the following scenarios

```
// ReferenceError Exception: iAmUndeclared
if ( iAmUndeclared < 5 ) {

}
```

Comparing to undefined is no different from any other comparison and results in the same:

```
// ReferenceError Exception: iAmUndeclared
if ( iAmUndeclared === undefined ) {

}
```

iAmUndeclared was either intended to be a local variable not declared through var, or a global variable that for some reason isn't available. Either way a bad situation that can be prevented:

Local variable

```
( function() {
  var foo, bar, baz;

  if ( somethingTrueIsh() ) {
    bar = 1024;
  }

  // Does NOT throw a ReferenceError Exception,
  // foo is undefined but not undeclared, it simply returns true
  if ( foo === undefined ) {
    // ...
  }

  if ( foo < 5 ) {
    // ...
  }

} );
```

Global variable

```
// Does NOT throw a ReferenceError Exception,
// No need for typeof
if ( window.globalVariableThatDoesNotExist ) {
  // ..
}
```

There is one more common scenario I'd like to touch here and that's variable declared/defined conditionally. This is a bad habit, because it's confusing. Consider the following construction:

```
foo.localScoopy = function() {
  /**
   * Imagine a complex way to determine some value for 'meh'.
   */
  if ( mw.isAwesome() ) {
    var meh = 'whee';
  }
}
```

```

}
if ( typeof meh === "undefined" && mw.isCrazy() ) {
  var meh = 'wokkel';
}
if ( typeof meh === "undefined" ) {
  // .. etc

```

It looks like one has to use `typeof` here because `meh` could be undeclared, since the `var` is inside the condition. So would this really be a situation in which we need `typeof` ? No. Using strict comparison instead of `typeof` would actually work just fine in the above example. This is because "var" statements, in JavaScript, are "hoisted" internally to the top of the current scope (regardless of whether it's declared conditionally). More about hoisting of variables and functions: [nettuts+ \[2\]](#), [adequatelygood.com \[3\]](#).

As a good practice, it is recommended to combine all your `var` statements at the beginning of a scope. Our example would then look like this:

```

/**
 * Imagine a complex way to determine some value for 'meh'.
 */
foo.localScoop = function() {
  var meh;

  if ( mw.isAwesome() ) {
    meh = 'whee';
  }
  if ( meh === undefined && mw.isCrazy() ) {
    meh = 'wokkel';
  }
  if ( meh === undefined ) {
    // .. etc

```

Fallback

Lastly, to put these cases in a real-world scenario, let's provide default values / fallbacks for variables, in a way that doesn't need a `typeof` check either.

```

function bar( lorem ) {
  lorem = lorem === undefined ? 'default' : lorem;
}

```

Or even, if your argument won't / shouldn't be false-y:

```

function bar( lorem ) {
  lorem = lorem || 'default';
}

```

Null OR undefined

Checking if a variable is one of 'null' or 'undefined' (like `isset()` in PHP) can be done comparing to both.

```

function bar( lorem ) {
  if ( lorem === undefined || lorem === null ) {
    // lorem was not set (undefined) or is set to null or undefined.
  } else {

```

```
    // lorem was 'set'.  
  }  
}
```

Array detection

```
var foo = ['zero', 'one', 'two'];  
  
typeof foo === 'array'; // false
```

Arrays do not have a `typeof "array"`. Nor do they validate as `"instanceof Array"` (atleast not consistently and cross-browser).

As of JavaScript 1.8.5, the array object constructor has an native object method (not a prototype) called `"isArray"`. It can be used like this:

```
var bar = ['cero', 'uno', 'dos'];  
  
Array.isArray( bar ); // true
```

However this is new and not cross-browser yet. The other reliable way to do this to call `".toString"` and verify that it is equal to `"[object Array]"`. However this is slightly slower.

jQuery has had a built-in object method called `"isArray"` that does the latter `isString()` comparison that works cross-browser. As of jQuery 1.4 (up to atleast 1.6.1) it only does that if the, faster, native `Array.isArray` isn't available. The best of both.

So the safest and fastest way to check if a variable is an array is to use `jQuery.isArray`.^[4]

String concatenation

As in a number of languages, concatenating an array of strings together at once is often faster than concatenating them one at a time, as it avoids creating a lot of intermediate string objects that will never get used again.

In inner loops this sometimes makes a significant time difference -- usually it will be far too small to measure for any one-off operation.

<http://jsperf.com/build-or-regex-array-vs-string>

```
var a = [];  
a.push(foo);  
a.push(bar);  
a.push(baz);  
  
var b = a.join('|'); // ["foo", "bar", "baz"] => "foo|bar|baz"  
  
var a = '';  
a += foo + '|';  
a += bar + '|';  
a += baz + '|';  
  
var b = a.slice(0, -1); // "foo|bar|baz|" => "foo|bar|baz"
```

WikiEditor

Some stuff Catrope discovered when optimizing EditToolbar. These may not be universally true, so when in doubt run both alternatives through the Firebug profiler or use <http://jsperf.com>

- Building simple HTML that doesn't need stuff like `.data()` or `.click()` yourself and feeding it to `.html()` or `.append()` is literally about 100 times as fast (YMMV) as building it with jQuery
- Even if you do need `.data()` or `.click()`, something like this is still faster (at least for 20-200 children, YMMV):

```
var html = '';
html += 'build HTML here';
$( '#something' ).html( html ).children().click( function() {
    // whatever
} );
// or:
$( '#something' ).html( html ).children().each( function() {
    $(this).whatever();
} );
```

- Avoid setting `.data()` on hundreds of elements that have a common parent or near ancestor. Instead, add an empty object to the parent's `.data()` and fill that through a reference using the child's `rel` property as keys. This reduces the number of `.data()` calls a lot.
- `.text()` is faster^[5] than `.html()` if you're only adding text
- `.addClass('foo')` is faster^[6] than `.attr('class', 'foo')`

jQuery.size() vs .length property

As one can see in the jQuery source^[7], the `size`^[8] function returns the `.length` property of the jQuery object.

As there's no functional difference nor length in number of characters (`".size()"` vs. `".length"`) and only a small performance penalty^[9], its use is discouraged.

UI responsiveness

Try to keep heavy calculations/string processing, document reflows, and other slow manipulation to a minimum; keeping responsiveness good is important to keep the web site from feeling slow for users.

If you have to perform a lot of activity in response to some user action, consider dividing the processing into short chunks that can be split over timeouts or intervals, allowing UI events to be processed while you work. For some tasks, HTML 5 Web Workers may be appropriate, but we don't yet have good infrastructure for integrating these with ResourceLoader.

Startup

Initial page load is often the slowest part of things because we have to wait for a lot of resources to get loaded, be parsed by the browser, and get put together.

JavaScript that doesn't **need** to run at startup should consider delaying operations to avoid slowing things down.

In particular, try to avoid delaying loading by forcing a synchronous script load near the top of the page! This can actually delay seeing the entire page, which is very sad.

Animation

For a lot of common things like fade-in, fade-out, slide-up/down etc there are common jQuery functions set up to handle these for you.

In MediaWiki 1.18 and higher (jQuery 1.6 and higher) these will automatically make use of the `requestAnimationFrame`^[10] interface on supporting browsers, which can help the animations to run smoothly and use less CPU time.

It may also in some cases be more efficient to use CSS 3 transitions; however we don't use a lot that likely benefits from these yet.

Memory usage and leaks

Most of MediaWiki operates on a traditional page navigation/form-submission model; this means that you'll have the entire HTML & JavaScript state potentially wiped clean and rebuilt each time the user jumps from one page to another.

However, be aware that pages may be open for some time in a user's browser during editing, reading, etc -- or left open overnight. In 2011-2012 we can expect more ongoing interactions and real-time updates to happen without actually switching pages, and so being aware of what gets held in memory and how long may become more important.

In particular, some things to watch out for:

- callbacks bound to HTML element events will stick around along with the elements -- including any variables and functions referenced from its closure context! In some cases you may want to discard an element or unbind a specific event handler after done with it.
- accidental globals stick around until the page closes; don't forget to use your 'var's

Notes

[1] <https://developer.mozilla.org/en/DOM/document.querySelector>

[2] <http://net.tutsplus.com/tutorials/javascript-ajax/quick-tip-javascript-hoisting-explained/>

[3] <http://www.adequatelygood.com/2010/2/JavaScript-Scoping-and-Hoisting>

[4] Testing Array-type Speed Performance comparison (<http://jsperf.com/testing-array-type-speed-performance-comparison>) on jsPerf.com

[5] jQuery `text()` vs. `html()` (<http://jsperf.com/jquery-text-vs-html/2>) on jsPerf.com

[6] jQuery `addClass(...)` vs `attr('class', ...)` (<http://jsperf.com/jquery-addclass-vs-attr-class>) on jsPerf.com

[7] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/jquery/jquery.js?view=markup#l206>

[8] <http://api.jquery.com/size/>

[9] jQuery `.size()` vs `.length` (<http://jsperf.com/size-vs-length>) on jsPerf.com

[10] <https://developer.mozilla.org/en/DOM/window.mozRequestAnimationFrame>

jQuery

MediaWiki	jQuery	Notes
MediaWiki 1.16	jQuery 1.3.2	
MediaWiki 1.17	jQuery 1.4.2	r65962
MediaWiki 1.18	jQuery 1.6.4	r92349, bug 28904, bug 29773 r99262
MediaWiki 1.19	<i>jQuery 1.6.4</i>	r91845, r99231
MediaWiki 1.20	<i>Unknown</i>	bug 29100,

We have been shipping the **jQuery** JavaScript library with MediaWiki since the 1.16 release.

ResourceLoader

MediaWiki version:	≥ 1.17
--------------------	-------------------------------

As of MediaWiki 1.17 all resources are (or should be) loaded through ResourceLoader. The default modules are stored in `/resources`. The jQuery file is in `/resources/jquery/jquery.js`. There are no static minified versions anymore as ResourceLoader takes care of this when combing and optimizing all queued files.

Before ResourceLoader

MediaWiki version:	≤ 1.16
--------------------	-------------------------------

MediaWiki 1.16 shipped with jQuery version 1.3.2 with some patches by Trevor Parscal and Roan Kattouw to fix jQuery bug #5081^[1] (see r60697 and r61700).

In 1.16, the jQuery file is in `skins/common/jquery.js` and the minified version is `jquery-min.js` in the same directory.

Extensions can use OutputPage's `includejQuery` method, which has existed since r61431, to avoid loading minified version of jQuery core multiple times. It should be noted that `includejQuery` does not support jQuery plugins and such.

To include it, paste the following at the bottom of `LocalSettings.php`

```
// Include jQuery
function wfIncludejQuery() {
    global $wgOut;
    $wgOut->includejQuery();
}
$wgExtensionFunctions[] = 'wfIncludejQuery';
```


1.16 jQuery.noConflict

Please pay attention to the fact that, to avoid conflicts with other libraries (since it's not loaded by default in 1.16) jQuery will be loaded in noConflict() ^[2]-mode into `window.$j`.

You can use either `jQuery`, `$j` or something like the following to locally wrap-alias it:

```
( function( $ ) {  
    // Simple scope wrap, where $ is available as alias for jQuery.  
    // Code here will be executed immediately  
} )( jQuery );  
  
/* or */  
  
jQuery( document ).ready( function( $ ) {  
    // Code wrapped in this function will be executed when the  
document is ready  
    // When the document is ready, jQuery calls this passed  
callback function like this:  
    // callbackFunction( jQuery );  
});
```

External links

- [jQuery.com](http://jquery.com), the official jQuery site ^[3]
- [jQuery Dialog error in IE7. Runtime error: invalid argument - The Dev Pages](http://www.n8williams.com/devblog/javascript_and_ajax/jquery-dialog-error-in-ie7-runtime-error-invalid-argument) ^[4] — the reason why we patched our jQuery

References

- [1] <http://dev.jquery.com/ticket/5081>
[2] <http://api.jquery.com/jquery.noConflict/>
[3] <http://jquery.com>
[4] http://www.n8williams.com/devblog/javascript_and_ajax/jquery-dialog-error-in-ie7-runtime-error-invalid-argument

Pre-commit checklist

This is an attempt to create a checklist to use before making a commit. Some of this duplicates what is in the coding conventions, but is in the form of quick checks. This checklist is in the same vein as The Checklist Manifesto ^[1]. Some of these may seem silly (like asking a doctor “did you wash your hands?”) but they’re meant to avoid problems that may be overlooked.

- Did your code run **without errors under E_STRICT**?^[2]
- Did your code break any of the unit tests? See Manual:PHP unit testing and Manual:JavaScript unit testing
- Have you tested all exit points from your code?
- Did you use **tabs** instead of spaces to indent?
- If you've created a new function, did you **document the function** parameters and what it returns using ?
- **Whitespace trailing** at the end of lines is annoying, don't do that (but don't fix whitespace at the end of lines you didn't modify in the same commit, just make sure whatever you touched is good, see next point:)
- Does this commit contain any whitespace-only changes? They should be isolated to their own **separate coding style-only commit**.
- Have you created any identifiers that didn't use **camelCase** (ie. underscores)?
- Is every exception tested?
- If you have multiple return points, are they tested?
- Does each message you've created exist in `MessagesEn.php` and listed in `maintenance/languages/messages.inc`?
- Is each use of `fopen()`, `fread()`, etc. checked for errors or problems?
- Did you use `t` or `b` flags for `fopen()` to ensure Windows compatibility?
- Have you used the proper output functions? `echo` should almost never be used.
- Have you used the proper termination functions? `exit` should almost never be used.
- Where appropriate, have you used the MediaWiki wrapper functions instead of their PHP equivalents?
 - `wfIniGetBool()` instead of `ini_get` to get boolean params
- If you add a new test to `parserTests.txt` did you give it a new name?
- Have you run `php checkSyntax.php --modified`?

JavaScript /CSS

- Tested in an actual browser?
- Compatible with IE 6.0+, FF 2.0+, Safari 3.0+, Opera 9.0+, Chrome?
- (JavaScript) Does it validate (or did you at least run it through) JSHint ^{[36][3]} ?
- (JavaScript) Are there any implied globals other than `jQuery` or `mw`? There should not be, (not `$` either)
- Is it DRY?

[1] <http://www.amazon.com/Checklist-Manifesto-How-Things-Right/dp/0805091742>

[2] Put `error_reporting` (<http://php.net/manual/en/function.error-reporting.php>) `(-1)` ; in the entry file. See also Manual:How to debug

[3] Recommended JSHint settings: [x] Require curly braces around all blocks; [x] Don't check line breaks; [x] Browser

How to debug

This page gives a basic introduction to **debugging MediaWiki** software.

One of the first things you will notice is that "echo" generally does not work; this is part of the general design.

PHP errors

To see PHP errors, add this to the very top of `LocalSettings.php`:

```
error_reporting( E_ALL );
ini_set( 'display_errors', 1 );
```

This will cause PHP errors to be shown on-page. This might make it easier for attackers to find a way into your server, so disable it again when you have found the problem.

Note that *fatal* PHP errors may happen before the lines above are ever executed, or may prevent them from being shown. Fatal PHP errors are usually logged to Apache's error log – check the `error_log`^[1] setting in `php.ini` (or use `phpinfo()`^[2]).

Any of the `$wg` type settings should be added to the bottom of your `LocalSettings.php`.

You can enable more details (like a stack trace) to be shown for some types of errors:

```
$wgShowExceptionDetails = true;
```

This is especially useful when debugging errors in the PHP code (as opposed to configuration problems).

Note that `E_ALL` does not include `E_STRICT`, so if you want to receive notices, you may need to use instead:

```
error_reporting( E_ALL | E_STRICT );
ini_set( 'display_errors', 1 );
```

You may want to set, in `php.ini`:

```
error_reporting = E_ALL | E_STRICT
```

SQL errors

To display SQL errors in error messages instead of "(SQL query hidden)", add the following to `LocalSettings.php`:

```
$wgShowSQLErrors = true;
$wgDebugDumpSql  = true;
```

In-depth debugging

Logging

For much greater detail, you need to profile and log errors.

To save SQL errors to a log, add `$wgDebugLogFile` to the `LocalSettings.php` file after the invocation of `DefaultSettings.php`. Change the value to a text file where you want to save the debug trace output.

```
/**
 * The debug log file should be not be publicly accessible if it is
used, as it
 * may contain private data. But it must be in a directory writeable by
the
 * PHP script runing within your Web server. */
$wgDebugLogFile = 'C:/Program Files/Apache
Group/htdocs/mediawiki/debug_log.txt';
```

This file will contain all of the built-in MediaWiki debug information as well as anything you try to log. To create a custom log file that only holds your debug statements, add this to `LocalSettings.php`.

```
/**
 * The debug log file should be not be publicly accessible if it is
used, as it
 * may contain private data. But it must be in a directory to which PHP
run
 * within your Web server can write. */
$wgDebugLogGroups = array(
    'myextension'      => 'logs/myextension.log',
    'anotherloggroup' => 'logs/another.log',
);
```

Then debug to this custom log using a statement like this:

```
// ...somewhere in your code
if ( $bad ) {
    wfDebugLog( 'myextension', 'Something is not right: ' . print_r(
$editpage->mArticle, true ) );
}
```

Profiling

To get more detail, you need to enable profiling. Profiling tracks code execution during a page action and reports back the percentage of total code execution that was spent in any specific function. The generated profile only includes functions that have specifically been marked to be profiled.

MediaWiki version:	≥ 1.8
--------------------	--------------

To enable profiling, you need to modify the `StartProfiler.php` (see `StartProfiler.sample` in the MediaWiki root folder for an example). By default the file includes a `ProfilerStub` which just dumps profiling information. To instead direct this information to a file, edit `StartProfiler.php` so that it looks like this:

```
#require_once( dirname( __FILE__ ) . '/includes/ProfilerStub.php' );

require_once(  dirname( __FILE__ ) . '/includes/Profiler.php' );
$wgProfiler = new Profiler;
```

Then you can customize profiling options `LocalSettings.php` (not `StartProfiler.php`; be sure to edit beneath the requirement of `DefaultSettings.php`).

Common configuration (both <1.7 and >1.8):

```
// Only record profiling info for pages that took longer than this
$wgProfileLimit = 0.0;
// Don't put non-profiling info into log file
$wgProfileOnly = false;
// Log sums from profiling into "profiling" table in db
$wgProfileToDatabase = false;
// If true, print a raw call tree instead of per-function report
$wgProfileCallTree = false;
// Should application server host be put into profiling table
$wgProfilePerHost = false;

// Settings for UDP profiler
$wgUDPProfilerHost = '127.0.0.1';
$wgUDPProfilerPort = '3811';

// Detects non-matching wfProfileIn/wfProfileOut calls
$wgDebugProfiling = false;
// Output debug message on every wfProfileIn/wfProfileOut
$wgDebugFunctionEntry = 0;
// Lots of debugging output from SquidUpdate.php
$wgDebugSquid = false;
```

MediaWiki version:	≤ 1.7
--------------------	--------------

In MediaWiki 1.7 and earlier, instead of editing `StartProfiler.php`, you have to set `$wgProfiling` to `true`. This will generate basic page timing information in the file defined by `$wgDebugLogFile`.

In addition to the settings list above, these additional settings are available:

```
// Enable for more detailed by-function times in debug log
$wgProfiling = true;
// Only profile every n requests when profiling is turned on
$wgProfileSampleRate = 1;
// If not empty, specifies profiler type to load
$wgProfilerType = '';
```

Advanced profiling

Once you have enabled profiling, you can trace code execution through any function that you want to investigate as a bottleneck by wrapping the function with the following code:

```
function doSomething() {
    wfProfileIn( __METHOD__ ); # You can replace __METHOD__ with any
    string. This will appear in the profile.

    # The actual function

    wfProfileOut( __METHOD__ );
}
```

After you've added this information, browse to a page in the wiki. This will generate profiling info in the log file you defined above. Change `$wgProfileCallTree` in `LocalSettings.php` to true or false for different display formats.

Logging to Database

To log profiling information to a database, first you'll have to create a profiling table in your MediaWiki database the table definition in the file `maintenance/archives/patch-profiling.sql` (the recommended way to do this is `php maintenance/patchSql.php profiling`). Then set `$wgProfileToDatabase = true;` in `LocalSettings.php`.

Note: `$wgProfileCallTree` *must* be set to false.

Viewing Profile Info

If you log your profiling information to the database, you can view the information in a webpage by browsing to `profileinfo.php`. You must also set `$wgEnableProfileInfo = true;` in `AdminSettings.php`. Then, after gathering data by browsing wiki pages, visit `profileinfo.php` to see how much time your profiled code is using and how many times it's being called.

To view profiling information as HTML comments appended to the bottom of a page, just add `?forceprofile=true` to the URL. This feature is not in the standard product, you can enable it by adding this to `StartProfiler.php`:

```
if ( array_key_exists( 'forceprofile', $_REQUEST ) ) {
    require_once( dirname( __FILE__ ) .
'/includes/profiler/ProfilerSimpleText.php' );
    $wgProfiler = new ProfilerSimpleText;
    $wgProfiler->setProfileID( 'forced' );
} elseif ( array_key_exists( 'forcetrace', $_REQUEST ) ) {
    require_once( dirname( __FILE__ ) .
'/includes/profiler/ProfilerSimpleTrace.php' );
    $wgProfiler = new ProfilerSimpleTrace;
}
```

Useful debugging functions

- `wfDebugDieBacktrace()` ^[3] – dies with an optional message and prints the call-stack

Debugging certain functionality

- Resource Loader: ResourceLoader/Developing with ResourceLoader

References

[1] http://php.net/error_log

[2] <http://php.net/phpinfo>

[3] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#2afcb6f3b8c61b2a695c8e4528d1ce40

Requests for comment page: discuss your new idea before committing

shortcut RFC

This is a **requests for comment** system that allows MediaWiki developers and users to draft large ideas for the future of MediaWiki.

In draft

Feature	Status
Partial page caching	2011-11-23 — RFC page and Bugzilla bug created
Notification framework	2011-11-08 — Add a common framework for storing/sending/reading talk, watch, and other notifications
Internationalization testing	2011-10-10 — Add a framework for internationalization testing
Refactor on File-FileRepo-MediaHandler	2011-10-03 — Refactor on File-FileRepo-MediaHandler
Decorify into extensions	2011-09-26 — Move core features into extensions
Drop actions in favour of page views and special pages	2011-08-26 — Replace Article and Actions with a PageView system and SpecialPages for actions
Itemise protection	2011-08-04 — Itemise page restrictions to allow for "surges" in protection whilst retaining fallback
Reimplement info action	2011-05-17 — Making ?action=info super-useful
Extension release management	2011-03-24 — Initial draft, building on ideas from 2010 GSoC
Account creation	2011-03-09 — Improving the account creation interface
Configuration database	2011-01-30 — How to revamp our configuration management interface and provide UI tools for wiki administrators
Distribution and deployment	2010-06-01 — Create systems to update MediaWiki core and extensions from a SpecialPage, and push updates through a distribution channel from mediawiki.org
Page deletion	2010-05-10 — How to apply RevDelete to page deletion?
Nasty templates	2010-05-04 — implementation ideas being floated

In discussion

Feature	Status
Database field for checksum of page text	2011-08-18 — In discussion on wikitech-l ^[1]
Unit testing	2011-07-05 — In discussion on wikitech-l ^[2]

Accepted

Feature	Status
Reduce math rendering preferences	2011-11-28 — Options eliminated; still need to add baseline shift

Implemented

Feature	Status
Context object	2011-02-11 — Replace all request-dependant objects members with only one "context" object

On hold / outdated

Feature	Status
Extensionless files	2010-04-02 — In discussion on wikitech-l and at bugzilla:4421

References

[1] <http://lists.wikimedia.org/pipermail/wikitech-l/2011-August/054772.html>

[2] <http://lists.wikimedia.org/pipermail/wikitech-l/2011-July/054160.html>

Code review

Code review

Code review is the systematic examination of MediaWiki core and extension code revisions intended to find and fix mistakes overlooked in development. Please test all your code before committing and use a descriptive commit message to make it easier for code reviewers to understand the reasoning behind your revision and to know that all the changes you made were intentional.


Extension:CodeReview helps facilitate the code review process.

Status flags

CodeReview allows revisions to be marked as:

Code review status flags

Status flag	Meaning
new	This code is in need of review. This is applied to new revisions and should also be applied when all the known issues with a FIXME revision have been satisfactorily addressed, but a full review of the revision has still not yet been done.
fixme	A reviewer has identified a specific problem with the revision that needs to be fixed, and no attempt to fix it has been reviewed, or the attempt to fix it has been reviewed and deemed unsatisfactory.
reverted	The revision has been reverted in a later revision.
resolved	After issues with the revision were noted and satisfactorily addressed, a full review of the code has been done and the reviewer is sure that the revision is OK in every way.
ok	A full review of the revision has been done and the reviewer is sure that the revision is OK in every way.
deferred	Commonly used for extensions that are not enabled on Wikimedia wikis because many of the reviewing developers are focused there, so from their perspective, review isn't strictly necessary. If your code is marked "deferred" and you were hoping for a review, just ask (for example, check in on the #mediawiki IRC channel and/or reopen as "new" with a comment asking for review.)
old	Revisions older than a certain age (generally speaking, r50000), not explicitly marked not new/reviewed.

 **Warning:** Please do not change the flags for your own revisions, except to change them back to "new" after you fix a revision that was flagged "fixme." All developers are encouraged to fix "fixme" status revisions regardless of who made the revision.

Emails may be sent out periodically via Wikitech-l notifying everyone of how many outstanding "fixme" commits each committer has.

Magic linking

Syntax like

- bug 12345
- r12345
- follow up patch to r12345

in a commit message or code comment automatically creates a link to the specific bug or revision, or set a follow-up indication at the specific revision it follows up. When used in a commit message specifically, this syntax also creates links between revisions (to note follow-up revisions, for example).

The commit message and code comment parsers are very finicky. This is noted at bug 24279.

External links

- A bug's life cycle ^[1]

References

[1] <https://bugzilla.wikimedia.org/page.cgi?id=fields.html#status>

Code review guide

This is a guide to reviewing contributions to the MediaWiki codebase. While this guide is written primarily for MediaWiki developers doing code reviews, developers submitting code for review will also find the guide useful.

Goals

- Encourage new developers to continue contributing to MediaWiki.
 - Be nice. Frame your comments as helpful suggestions.
 - Tolerate idiosyncrasies where possible. Developers should feel that their creative input is valued and preserved.
 - Review patches and commits soon after they've been contributed, to avoid bitrot and feelings of abandonment.
- Encourage community developers to contribute to the code under review.
 - Ensure that the code is readable and follows the coding conventions.
- Contributed code should work as advertised, that is, any bugs found in the code should be fixed.
- Protect MediaWiki users by ensuring an absence of security vulnerabilities. The code should follow the best practices outlined in Security for developers.
- Aim for interface stability and backwards compatibility.
 - Review public interfaces carefully, so that future changes can be kept to a minimum.

Choosing revisions

There is a lot of code to review. How do we break up the task into more manageable portions?

There are several basic patterns which reviewers have found to be useful:

Chronological (and Reverse Chronological)

- Start at the earliest revision, read each diff in turn until you get to HEAD. Alternately, start at the latest revision and read each diff in turn until you reach reviewed code.
- This approach is good for minor revisions, but using it exclusively has a negative effect on reviewer sanity.
- The way different projects are interleaved in the chronological view means that it feels like you're constantly suspending one project, doing something else for a while, then resuming where you left off.
- When reviewing chronologically, it's common to erroneously flag a revision as "fixme" when it's already been fixed in a later revision, which you haven't seen yet.
- When reviewing in reverse chronological order, it is easier to spot revisions which have been superseded.

By project

- Identify major pieces of work or sets of related revisions. A quick chronological review can be a good way to do this.
- Open all Code Review revision pages as tabs in a browser window. Open relevant files in a text editor.
- Review new or unfamiliar files in their entirety. This is an engaging and satisfying way to work, and so should be preferred when possible. Pick a revision with a major change to the relevant file and put your comments on that Code Review page.
- If you review a file completely, then it's possible to rapidly mark changes to that file before the review point as "ok" or "deferred" (for superseded revisions).
- Be careful not to blame the current developer for code written by a previous developer. Use the `svn annotate` command to trace changes to the revision where they were introduced. You can ask the current developer to fix the previous developer's code, but be aware that they will not feel obliged to do so, so frame your request accordingly.
- If a project involves a major change to a large file, and the change is spread over many revisions, you can use `svn diff` to combine all the changes into a single file. This allows you to avoid reviewing superseded changes, and to get an overall view of the changes being made. This is faster than reviewing the revisions individually.
- If multiple projects are interleaved on the same file or directory, it may still be possible to get a project-specific diff, by checking out a new copy and using `svn merge` to grab the relevant changes. Then a plain `svn diff` gives you the combined changes.
- Edit this Bugzilla query ^[1] to only give bugs in a particular MediaWiki component, such as API or Page editing.
- This method does not work for minor changes that are not part of any project. It also has some overhead: it requires the reviewer to identify projects and changesets. In some cases this is easy (such as when reviewing a new extension), but at other times it is less obvious.

By author

- Make a list of authors, pick one, load their revision list in CodeReview.
- Work through the revisions chronologically, or split their work up into projects and review by project.
- This method allows the reviewer to get to know individual developers: their skills, faults and interests. The work has a sense of progress and continuity.

Randomly

- Copy the code from User:Bryan/vector.js that adds a random button to Special:Code
- Click it and start reviewing a random revision.
- This method provides the excitement Special:Random
- Or, choose a patch that needs reviewing at random from the Bugzilla list ^[1].

Review responses



Warning: Please do not change the flags for your own revisions, except to change them back to "new" after you fix a revision that was flagged "fixme." All developers are encouraged to fix "fixme" status revisions regardless of who made the revision.

- Good revisions should be marked "ok". **If you are particularly impressed by someone's work, say so in a comment.**
- Trivial revisions which are broken and have no redeeming value should be reverted on the spot. Do not write "please revert this immediately" in a comment, such requests are almost never complied with and are easy to misunderstand.
- Revisions which have issues but also have some utility, or are showing signs of a developer heading in the right direction, can be given a comment and a "fixme" status. (If someone marks your code fixme, this doesn't mean that your code sucks - it means that it's good, but needs improvement.)
- Revisions which are bad but have been fixed in a subsequent commit can be marked "ok" or "resolved". Note that code flagged as "ok" often won't be looked at - be very careful marking things as ok.
- Revisions which have been reverted should be marked "reverted".
- Reversions themselves are always "ok".
- The following kinds of changes should be marked "deferred":
 - i18n updates. We don't mark them "ok" because we don't review the diffs.
 - Changes to extensions or other software which does not run on Wikimedia, with the possible exception of Semantic MediaWiki.
 - Changes to private or development branches.
 - Very old changes made by a reviewer, where there is no hope that someone else will review their changes.
- A backport to a release branch can be marked "ok" if it has suitable release notes, if the case for backporting was sufficiently strong, and if the trunk equivalent was marked "ok".
- In most cases, a backport to a deployment branch can be marked "ok", on the basis that people who have access to the deployment branches know what they are doing. It can be "ok" even if the trunk equivalent is marked "fixme", since temporary hacks are acceptable in deployment branches.
- A revision which should be backported to a branch should be tagged with the branch name, for instance "1.17" or "1.16wmf4". Once the backport is done, the tag should be removed.
- A revision which was committed to a branch and needs to be ported to trunk should be tagged with "trunk".
- Schema changes should be tagged "schema".
- Changes which may cause problems when they are deployed to Wikimedia should be tagged as "scaptrap". This includes:
 - Most schema changes.
 - Changes which require a simultaneous update to the configuration files.
 - Changes which put significant pressure on server capacity, such as increases in request rate, outgoing bandwidth or disk space.
 - Changes to extensions which rely on changes to core MediaWiki that have not been reviewed and merged

Code review status flags

Status flag	Meaning
new	This code is in need of review. This is applied to new revisions and should also be applied when all the known issues with a FIXME revision have been satisfactorily addressed, but a full review of the revision has still not yet been done.
fixme	A reviewer has identified a specific problem with the revision that needs to be fixed, and no attempt to fix it has been reviewed, or the attempt to fix it has been reviewed and deemed unsatisfactory.
reverted	The revision has been reverted in a later revision.
resolved	After issues with the revision were noted and satisfactorily addressed, a full review of the code has been done and the reviewer is sure that the revision is OK in every way.
ok	A full review of the revision has been done and the reviewer is sure that the revision is OK in every way.
deferred	Commonly used for extensions that are not enabled on Wikimedia wikis because many of the reviewing developers are focused there, so from their perspective, review isn't strictly necessary. If your code is marked "deferred" and you were hoping for a review, just ask (for example, check in on the #mediawiki IRC channel and/or reopen as "new" with a comment asking for review.)
old	Revisions older than a certain age (generally speaking, r50000), not explicitly marked not new/reviewed.



Warning: Please do not change the flags for your own revisions, except to change them back to "new" after you fix a revision that was flagged "fixme." All developers are encouraged to fix "fixme" status revisions regardless of who made the revision.

The following table shows appropriate actions to take for a given review situation:

Code review response cheatsheet

Revision is in or is ...	Action to take
General	
Good	Mark "ok"
Trivial & broken	Revert revision
Useful but flawed	Mark "fixme"; Comment
Bad but fixed	Mark "resolved"
Reverted	Mark "reverted"
Reversion	Mark "OK"
Deferring	
i18n updates	Mark deferred
extension not used by Wikimedia	Mark deferred
Private branch	Mark deferred
Development branch	Mark deferred
Very old	Mark old
Back port and merge	
Backport to release branch	If it has a RELEASE-NOTES entry. If its trunk revision equivalent is OK Then mark OK.
Backport to deployment branch	Marked OK most of the time.
Need backport	Tag branch name

Backported	Remove branch name tag
Need port to trunk	Tag "trunk"
Change to schema	Tag "schema"
Wikimedia deployment	
Schema change	Tag "scaptrap"
Requires configuration file update	Tag "scaptrap"
Impacts server capacity: <ul style="list-style-type: none"> request rate outgoing bandwidth disk space ... 	Tag "scaptrap"
Extension change depending on unmerged core change	Tag "scaptrap"

Sign-offs

There is also a sign-off feature, that allows you to specify that you've *tested* or *inspected* the code, even if you do not feel confident enough to mark the code as being *ok*. All changes are logged.

Sign-offs

	User	Flag	Date
Strike selected sign-offs			Sign off on this revision as: <input type="checkbox"/> Inspected <input type="checkbox"/> Tested Sign off

Follow-up revisions

Rev.	Commit summary	Author	Date
Remove selected associations		Associate follow-up revision: <input type="text"/>	Associate

Example for a revision with the flag "Tested" (which could be made unchecked later, if needed)

Sign-offs

	User	Flag	Date
<input type="checkbox"/>	Wikinaut	tested	21:54, 24 May 2011
Strike selected sign-offs			Sign off on this revision as: <input type="checkbox"/> Inspected <input checked="" type="checkbox"/> Tested Sign off

Follow-up revisions

Rev.	Commit summary	Author	Date
Remove selected associations		Associate follow-up revision: <input type="text"/>	Associate

Dealing with non-compliance

It's quite common for a review comment with fixme status to go unanswered by the relevant developer. This occurs with both volunteers and staff.

Some amount of nagging is probably fair and useful, and if the developer is an employee, you have the option of getting their supervisor to help with that. But with some people, at some point it will come down to a choice of three options:

1. Revert
2. Tolerate the fault
3. Fix it yourself

You have to weigh up the costs and benefits of each course of action, they are all valid. If you revert, then that feature will be lost, and the developer may be discouraged. If you tolerate the fault, the end product will suffer, as described in the goals section above. If you fix it yourself, then you're letting yourself get distracted from the review task, you're letting the non-compliant developer dictate your priorities, and perhaps you're sending the wrong message to the developer, who may believe that it is acceptable to commit low-quality code and then wait for someone else to fix it.

In cases where fixing it makes sense, say because the feature is too useful or the bug fix too important to allow reversion, then consider whether it is possible to find someone else to fix it instead, so that your review work can continue uninterrupted.

How to review a change

Style

- Ensure the style guide is followed, such as whitespace, line length, brace placement, etc. This is easy to check. It is essential for core contributions. For extension contributions, style deviations can be tolerated initially.

Readability

- Functions should do what their names say. Choosing the correct verb is essential, a `get*()` function should get something, a `set*()` function should set something.
- Variables names should use English, abbreviations should be avoided where possible.
- Doc comments on functions are preferred.
- Overly-complicated code should be simplified. This usually means making it longer. For instance:
 - Ternary operators (`?:`) may need to be replaced with `if/else`.
 - Long expressions may need to be broken up into several statements.
 - Clever use of operator precedence, shortcut evaluation, assignment expressions, etc. may need to be rewritten.
- Duplication should be avoided.
 - Duplication, whether within files or across files, is bad for readability since it's necessary to play "spot the difference" to work out what has changed. Reading many near-copies of some code necessarily takes longer than reading a single abstraction.
 - It's also bad for maintainability, since if a bug (or missing feature) is found in the copied code, all instances of it have to be fixed.
 - It's common for new developers to copy large sections of code from other extensions or from the core, and to change some minor details in it. If a developer seems to be writing code which is "too good" for their level of experience, try grepping the code base for some code fragments, to identify the source. Comparing the original and the copy allows you to get a true feeling for the developer's level of expertise. Then guide the developer towards either rewriting or refactoring, to avoid the duplication.

Security

- The reviewer should have read and understood the security guide and should be aware of the security issues discussed there.
- There should not be the remotest possibility of arbitrary server-side code execution. This means that there should be no `eval()` or `create_function()`, and no `/e` modifier on `preg_replace()`.
- Check for text protocol injection issues (XSS, SQL injection, etc.) Insist on output-side escaping.
- Check for `register_globals` issues, especially classic arbitrary inclusion vulnerabilities.
- Check any write actions for CSRF.
- Be wary of special entry points which may bypass the security provisions in `WebStart.php`.
- Be wary of unnecessary duplication of security-relevant MW core functionality, such as using `$_REQUEST` instead of `$wgRequest`, or escaping SQL with `addslashes()` instead of `$dbr->addQuotes()`.

Architecture

- Names which are in a shared (global) namespace should be prefixed (or otherwise specific to the extension in question) to avoid conflicts between extensions. This includes:
 - Global variables
 - Global functions
 - Class names
 - Message names
 - Table names
- The aim of modularity is to separate concerns. Modules should not depend on each other in unexpected or complex ways. The interfaces between modules should be as simple as possible without resorting to code duplication.

Logic

- Point out shortcuts and ask the developer to do a proper job.

Programmers have a model of what the code is meant to do in their heads. As long as you're a fast typist, programming is easiest when you just translate that model into code in the obvious way. But sometimes programmers think that it will be easier to write the code if it is shorter, so they figure out a new model which is almost the same as the old one, except it requires slightly less code. They think that it will do the same thing in all the cases they have in their heads, therefore it is good enough. This is a common source of bugs, since unexpected cases later come to light.

Taking shortcuts is counterproductive, since the amount of time spent figuring out the shortcut and verifying that it works could have been spent just typing out the original idea in full. The exception to this is if the developer is a very slow typist, but most of our developers can actually type pretty well.

- Read relevant bug reports or documentation.
 - Familiarise yourself with any relevant technical issues. Read relevant specifications or manual sections.
 - Schema changes or changes to high-traffic queries should be reviewed by a DB expert. Currently this means Domas or Tim.
 - "Hot" code, i.e. code that is run many times in a request, or code that is run on startup, should be reviewed for performance. Suspicious code may need to be benchmarked.
 - Any web-accessible code which is very inefficient (in time, memory, query count, etc.) should be flagged for a fix.
-

References

- [1] https://bugzilla.wikimedia.org/buglist.cgi?keywords=patch%2C%20need-review%2C%20&keywords_type=allwords&order=Last%20Changed&type0-0-6=matches&field0-0-4=short_desc&query_format=advanced&bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&product=MediaWiki&type0-0-2=substring&list_id=47471

Code review tags

Part of the Code review guide.

Code review tags

Tag	Meaning
General	
parser	Parser
frontend	CSS & JavaScript & jQuery
needs-php-test	PHPUnit
needs-js-test	QUnit
i18n	Internationalization & localisation
scaptrap	Gotchas for deployments and upgrades. See the code review guide.
design	This revision requires visual review aside from the code. Don't remove the tag after review. A revision is not OK until design review is done.
nodeploy	Commits in trunks, release branches or WMF used extensions that should not be deployed to the cluster till further discussion is reached.
Branching (see also Branch points)	
1.19revert	Remove from 1.19 trunk, re-do when trunk is 1.20+
1.18	Backport to branches/REL1_18
1.18wmf1	This revision should be pushed to the cluster ASAP.
post118deploy	This revision should be pushed to the cluster, but it can wait until the dust settles.
1.18revert	Needs to be reverted from branches/REL1_18
reverted1.18	Removed from 1.18, but may still need trunk review/fixme
1.18ok	DO NOT USE old fixmes that were deemed ok to deploy as part of 1.18, but should still be fixed for 1.19.
1.17wmf1	Merge from trunk to branches/wmf/1.17wmf1
1.17	Merge from trunk to branches/REL1_17
1.17revert	Remove from 1.17, add to/keep in 1.18+. Action:Undo in trunk now, merge the undo to REL1_17, Re-do in trunk (optionally tag re-do "1.18")
1.16	Merge from trunk to branches/REL1_16
live	Hack in a wmf-branch. Revert or merge to trunk.

Reviewer assignment

If a revision has been assigned to be reviewed by someone, it gets the tag of that reviewer's name.

Security

Security

This page documents information related to MediaWiki **security**.

Report a security problem

If you have found or believe you have found a security bug in MediaWiki or on one of Wikimedia's web sites, please directly e-mail **security@wikimedia.org** with details.

We would be most happy to have a day or two to fix the problem and prepare a bug fix for third-party users before public disclosure, if possible.

(Note that any security problems found in the wiki-to-HTML parser will be included in the parser regression test suite in the next release.)

Receive release notifications

You may subscribe to the low-traffic mediawiki-announce mailing list ^[1] to receive notifications of new MediaWiki releases by e-mail.

This will include all security fix releases as well as other new versions. Anyone running a MediaWiki installation is strongly recommended to subscribe.

References

[1] <http://mail.wikimedia.org/mailman/listinfo/mediawiki-announce>

Security for developers

As a MediaWiki developer, you have a responsibility to write secure code in a style that is easy to review and audit. This article focuses on the issues related to security and on the best practices used by MediaWiki developers to address these security issues. For issues of coding style, please read the MediaWiki coding conventions.

Every MediaWiki developer should carefully read this article, regardless of their level of experience in web application development and with PHP.

Why security matters

Web application security is a critical issue in the wired world. Websites with security vulnerabilities are a key part of the illicit global infrastructure of malware, spam and phishing. Bot herders crawl the web looking for websites with security vulnerabilities, and then use the vulnerabilities to hijack them. The hijacked website will distribute malware (viruses) to visitors, either via browser vulnerabilities or overtly by social engineering. The downloaded malware turns the client's computer into a "zombie" that is part of a global network of organized crime aimed at stealing bank account details, sending spam, and extorting money from websites with denial-of-service threats.

Demonstrable security

It's not enough to assure yourself that you are perfect and that your code has no security vulnerabilities. Everyone makes mistakes. All core code, and a good deal of extension code, is reviewed by experienced developers to verify its security. This is a good practice and should be encouraged.

Write code in such a way that it is **demonstrably secure**, such that a reviewer can more easily tell that it's secure. Don't write code that looks suspicious but is, on careful examination, secure. Such code causes unnecessary **reviewer anxiety**.

See the #Best practices section below for examples of code that is more secure.

Overview of security vulnerabilities and attacks

This document has a strong focus on the following attacks and security risks. Each MediaWiki developer should be familiar with these issues and have at least a passing understanding of them.

Cross-site scripting (XSS)

For detailed information on avoiding XSS vulnerabilities in MediaWiki, read the [Cross-site scripting article](#).

Cross-site scripting (XSS) vulnerabilities allow an attacker to inject malicious code into a website. XSS vulnerabilities are caused by a web application not properly escaping data from external sources (such as GET data, POST data, RSS feeds or URLs). The range of attacks that can be made via XSS are very diverse, ranging from harmless pranks to the hijacking of an authenticated user's account.

Primary defenses: To avoid XSS attacks, the basic principles are:

- Validate your input
- Escape your output

You can skip validation, but you can never skip escaping. Escape everything. Escape as close to the output as possible, so that the reviewer can easily verify that it was done.

Note that output encoding (escaping) is context sensitive. So be aware of the intended output context and encode appropriately (e.g. HTML entity, URL, Javascript, etc.)

The OWASP Abridged XSS Prevention Cheat Sheet ^[1] is a useful and up to date quick reference guide for mitigating XSS issues.

Cross-site request forgery (CSRF)

For detailed information on avoiding CSRF vulnerabilities in MediaWiki, read the Cross-site request forgery article.

Cross-site request forgery (CSRF or XSRF) attacks use authentication credentials cached in a victim's browser (such as a cookie or cached username and password) to authorize malicious HTTP requests. The malicious HTTP request can be sent in many ways. As long as the requests are processed by a web browser that has cached authentication credentials, a CSRF attack can be attempted.

Primary defenses: Our primary defense mechanism against CSRF attacks is to add edit tokens to HTML forms.

Register globals

For detailed information on avoiding variable injection when register globals is enabled, read the Register globals article.

`register_globals` is a deprecated configuration directive of PHP. When enabled, `register_globals` causes data passed to a PHP script via cookies or GET and POST requests to be made available as global variables in the script. This configuration directive is extremely dangerous, often allowing an attacker to overwrite variables in a script simply by adding parameters to requests.

Primary defenses: MediaWiki developers must write their code to defend against register globals-based variable injection, since `register_globals` may be enabled on servers where MediaWiki is installed. There are several guidelines to follow:

- Do not use global variables in script paths
- Make sure code is only executed in the right context (e.g. check that include files aren't being executed directly)
- Sanitize custom global variables before use
- Configure extensions only after their setup file is included

SQL injection

For detailed information on avoiding SQL injection, read the SQL injection article.

SQL injection relies on poorly validated input being used in a database query, possibly allowing an attacker to run arbitrary SQL queries on your server. The attacker may then be able to fetch private data, destroy data or cause other unintended responses. In the worst case, the injected code could allow the attacker to gain full control of the system by exploiting multiple vulnerabilities in the database server, system utilities and operating system.

Primary defenses: The primary defense against SQL injection is to use MediaWiki's built-in database functions. Avoid using direct SQL queries at all costs.

Best practices

If you are working with ...	have you ...
Cookies	<ul style="list-style-type: none"> reduced reviewer anxiety by using <code>\$wgRequest</code> instead of <code>\$_COOKIE</code>? <ul style="list-style-type: none"> fetches cookies using <code>\$wgRequest->getCookie(...)?</code> set cookies using <code>\$wgRequest->response()->setcookie(...)?</code> <pre># Attempt to fetch the UserID cookie value. Note: The # value returned isn't trusted and is forced to be an int. \$sId = intval(\$wgRequest->getCookie('UserID'));</pre>
Dynamic code generation	<p>Avoid using functions like <code>eval()</code> ^[2] and <code>create_function()</code> ^[3], as well as the <code>/e</code> pattern modifier ^[4] for <code>preg_replace()</code> ^[5]. While powerful and convenient, these features are inherently insecure: ^[6] ^[7]</p> <ul style="list-style-type: none"> it's easier to put arbitrary strings into text processed by a regular expressions, which – when combined with the <code>/e</code> pattern modifier – can lead to code injection attacks. it is harder to read and maintain code that is part of a string. static analysis tools won't catch warnings and errors in the code. opcode caches (like APC) can't cache code mixed into strings. <code>create_function()</code> sometimes has garbage-collection issues. A loop which has a <code>create_function()</code> inside will create a new function on each iteration. <p>Sometimes you really do need these features (obviously <code>eval.php</code> needs to run <code>eval()</code> ;) but in most cases, we'd rather see the function broken out and referred as a callback.</p> <p>For future code that runs only under PHP 5.3 and later, note that inline lambda functions will make it easier to make your callback inline while retaining the benefits of code that's written in native syntax instead of strings.</p> <ul style="list-style-type: none"> Anything external that is used in part of regex should be escaped with <code>preg_quote()</code> ^[8] (<code>\$externalStr, \$delimiter</code>). It puts a backslash in front of every character that is part of the regular expression syntax, and escapes also the delimiter given as second parameter: <pre>\$str = preg_replace("!" . preg_quote(\$externalStr, '!') . "!", \$replacement, \$str);</pre>
External programs	<ul style="list-style-type: none"> executed the program via <code>wfShellExec()</code> ^[9]? quoted all arguments to external programs using <code>wfEscapeShellArg()</code> ^[10]? <pre>// escape any naughty characters \$cmd = wfEscapeShellArg(\$cmd) . ' --version'; \$err = wfShellExec(\$cmd, \$retval); // run \$cmd</pre>
Forms	<ul style="list-style-type: none"> used <code>\$wgUser->editToken()</code> to implement anti-CSRF measures? reduced reviewer anxiety by using or extending MediaWiki's existing form functionality?
GET data	<ul style="list-style-type: none"> reduced reviewer anxiety by using <code>\$wgRequest</code> instead of <code>\$_GET</code>? <pre># Check if the action parameter is set to 'purge' if (\$wgRequest->getVal('action') == 'purge') { ... }</pre>
Global variables	<ul style="list-style-type: none"> written your code to defend against register globals-based variable injection? <pre># ensure that the script can't be executed outside of MediaWiki if (! defined('MEDIAWIKI')) { echo "Not a valid entry point"; exit(1); }</pre>

Output (API, CSS, JavaScript, HTML, XML, etc.) Any content that MediaWiki generates can be a vector for XSS attacks.	<ul style="list-style-type: none"> used the <code>Html</code> ^[11] and <code>Xml</code> ^[12] helper classes? <pre># rawElement() escapes all attribute values # (which, in this case, is provided by \$myClass) echo Html::rawElement('p', array('style' => \$myClass));</pre>
POST data	<ul style="list-style-type: none"> reduced reviewer anxiety by using <code>\$wgRequest</code> instead of <code>\$_POST</code>? <pre># Check if the action parameter is set to 'render' if (\$wgRequest->getVal('action') == 'render') { ... }</pre>
Query strings	<ul style="list-style-type: none"> See #Get data above
Sessions	
Reviewer anxiety	<ul style="list-style-type: none"> clearly commented unexpected or odd parts of your code? <pre># \$wgRequest isn't yet available. Forced to use \$_GET instead. if (isset(\$_GET['setupTestSuite'])) { \$setupTestSuiteName = \$_GET['setupTestSuite']; ... }</pre>
SQL queries	<ul style="list-style-type: none"> used MediaWiki's database wrappers?

Books

- Tobias Wassermann: "Sichere Webanwendungen mit PHP". ISBN 9783826617546

[1] https://www.owasp.org/index.php/Abridged_XSS_Prevention_Cheat_Sheet

[2] <http://php.net/eval>

[3] http://php.net/create_function

[4] <http://php.net/manual/en/reference.pcre.pattern.modifiers.php>

[5] http://php.net/preg_replace

[6] <http://www.mediawiki.org/wiki/Special:Code/MediaWiki/78046#c12250>

[7] https://bugzilla.wikimedia.org/show_bug.cgi?id=21526#c14

[8] http://www.php.net/preg_quote

[9] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#a8646a2e6a4961deb21639cfb8774cc82

[10] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#a781ca00c48d9c5cbd509c282a244c022

[11] <http://svn.wikimedia.org/doc/classHtml.html>

[12] <http://svn.wikimedia.org/doc/classXml.html>

Security (Manual)

If you believe you have found a security problem in MediaWiki or in one of Wikimedia's web sites, please contact security@wikimedia.org directly so we can prepare a bug fix release.

Stay up to date

The most important security step you can take is to keep your software up to date. Both MediaWiki and the software it depends on will occasionally produce new versions correcting newly-discovered security vulnerabilities which may affect you.

The MediaWiki developers strongly recommend that anyone running MediaWiki subscribe to the mediawiki-announce mailing list. This is a low-traffic list that sends out only new version announcements.

In accordance with MediaWiki's version lifecycle, each release will receive security updates for one year. Older releases may contain known-but-unpatched security holes.

Don't forget to also keep up with updates to Apache, PHP, MySQL, and any other software running on your servers -- both the operating system and other web applications. Several people's MediaWiki 1.3.x installations were affected in Fall 2004 by a worm attacking a flaw in phpBB; once it had gotten in through other customers' unpatched phpBB sites it added a "you are hacked" message to other writable .php files on the system, including the compiled templates that MediaWiki 1.3 used.

General PHP recommendations

These bits of advice go for pretty much any PHP environment, and are not necessarily specific to MediaWiki.

PHP configuration recommendations, for `php.ini` or set otherwise:

- Disable `register_globals` ^[1].
 - Many PHP security attacks are based on injection of global variable values, so making sure it's off can make many potential vulnerabilities toothless.
 - If you require `register_globals` for another web application, consider enabling it selectively, only for the virtual host or subdirectory that requires it.
 - MediaWiki should be safe even if this is on; turning this off is a precaution against the possibility of unknown vulnerabilities.
- Unless you require it specifically, disable `allow_url_fopen` ^[2].
 - Remote PHP code execution vulnerabilities may depend on being able to inject a URL into a `include()` or `require()`. If you don't require the use of remote file loading, turning this off can prevent attacks of this kind on vulnerable code.
 - MediaWiki may require this setting to be on for the Lucene search extension, the OAI harvester extension, and certain uses of `Special:Import` in 1.5. It should not however be required in a typical installation.
 - MediaWiki should be safe even if this is on; turning this off is a precaution against the possibility of unknown vulnerability.
- Set `session.use_trans_sid` ^[3] off.
 - If this is on, session IDs may be added to URLs sometimes if cookies aren't doing their thing. That can leak login session data to third-party sites through referer data or cut-and-paste of links.
 - You should always turn this off if it's on.

Your `php.ini` may be located in:

- `/etc/php.ini` (Red Hat Linux, SuSE / Novell Linux)
 - `/etc/php4/apache/php.ini` (Debian woody and sarge, Ubuntu 6.10 with php4 and apache 1.3)
-

- /etc/php5/apache2/php.ini (Ubuntu 6.10 with php5 and apache2, Debian etch)
- /etc/httpd/php.ini (Trustix Secure Linux 3.0)
- /usr/local/php/lib/php.ini (Mac OS X using Marc Liyanage's PHP package)
- /opt/local/etc/php5/php.ini (Mac OS X if using Macports)
- /Applications/MAMP/conf/php5/php.ini (Mac OS X 10.5/10.6 mit MAMP)
- /etc/apache/php.ini (Slackware 10.x), /etc/httpd/php.ini (on later Slackware versions)
- /var/www/conf/php.ini (OpenBSD)
- /usr/local/etc/php.ini (FreeBSD)
- /usr/pkg/etc/php.ini (NetBSD)
- /usr/syno/etc/php.ini (Synology NAS)
- Gentoo Linux:
 - /etc/php/apache2-php4/php.ini
 - /etc/php/cli-php4/php.ini
 - /etc/apache2/php.ini
- c:\windows\php.ini (Windows)

For instance if you see this line in php.ini:

```
register_globals = On
```

Change it to:

```
register_globals = Off
```

Alternatively, you could add this apache directive to turn off `register_globals` on a per-directory basis:

```
php_flag register_globals off
```

Then restart Apache to reload the changes by `apachectl reload` or `rcapache2 reload` (SuSE).

On a multiuser system with PHP installed as an Apache module, all users' scripts will run under the same reduced-privilege user account. This may give other users access to read your configuration files (including database passwords), read and modify your login session data, or write files into your upload directory (if enabled).

For multiuser security, consider using a CGI/FastCGI configuration in which each user's scripts run under their own account, or enabling Safe Mode ^[4] to limit script access to other users' files. Note that safe mode may interfere with some features of MediaWiki such as uploading and extensions which shell out to other programs.

General MySQL recommendations

In general, you should keep access to your MySQL database to a minimum. If it will only be used from the single machine it's running on, consider disabling networking support, or enabling local networking access only (via the loopback device, see below), so the server can only communicate with local clients over Unix domain sockets.

If it will be used over a network with a limited number of client machines, consider setting the IP firewall rules to accept access to TCP port 3306 (MySQL's port) only from those machines or only from your local subnet, and reject all accesses from the larger internet. This can help prevent accidentally opening access to your server due to some unknown flaw in MySQL, a mistakenly set overbroad GRANT, or a leaked password.

If you create a new MySQL user for MediaWiki through MediaWiki's installer, somewhat liberal access is granted to it to ensure that it will work from a second server as well as a local one. You might consider manually narrowing this or establishing the user account yourself with custom permissions from just the places you need. The database user only needs to have SELECT, INSERT, UPDATE and DELETE permissions for the database.[5]

Note that the `user` table in MediaWiki's database contains hashed user passwords and may contain user e-mail addresses, and should generally be considered private data.

See:

- mysql command-line options ^[6] `--skip-networking`.
- Setting `bind-address=127.0.0.1` in your `my.ini` (under section `[mysqld]`) will cause MySQL to only listen on the loopback interface. This is the default in the EasyPHP install for Windows. (If you are using MySQL on a Unix machine, the setting may be `skip-networking` instead in the `my.cnf` file.)
- GRANT and REVOKE syntax ^[7]

If the MySQL database has leaked

If the mysql database has leaked to the public, in `LocalSettings.php`:

1. Change `$wgDBpassword` if that leaked too
2. Change some letters in `$wgSecretKey`

Manual installation

If you use the web-based installer, you may be vulnerable to attack by somebody else running the installer on your server between the time you make the `config` directory writable and the time the `LocalSettings.php` file is written out.

Usually this should not be a very large risk, but if it's unacceptable to you (or if you need to do a batch install of many wikis), consider doing an installation manually:

- Create a database ^[8]
- Grant user permissions ^[7]. The database user only needs to have SELECT, INSERT, UPDATE and DELETE permissions for the database.^[5]
- Source `maintenance/tables.sql` to create the tables.
- Create a `LocalSettings.php` based on a sample or the generation code in `config/index.php`.

Note: The requirement for `AdminSettings.php` (along with supporting file `AdminSettings.sample`) was removed in MediaWiki 1.16. Prior to MediaWiki 1.16, `AdminSettings.php` was used to store database authentication credentials for maintenance scripts and to control the availability of `profileinfo.php`. Details can still be found in `Manual:AdminSettings.php`.

- `cd maintenance && php update.php`

If PHP is running as an Apache module, the `LocalSettings.php` generated by the web installer will usually be owned by the Apache user account. To ensure that it can't be changed again by another user (see notes above about multiuser systems) or by malicious code injected to a vulnerable web application, you should reassign it to another account. (If you have only limited access, consider copying instead of moving the file; the new copy will be under your other account.)

If LocalSettings.php has leaked

If LocalSettings.php has leaked to the public, reprotect it and:

1. Change \$wgDBpassword
2. Change some letters in \$wgSecretKey
3. Possibly make a slightly different \$wgSpamRegex

Database passwords

See Manual:Securing database passwords for some precautions you may wish to take to reduce the risk of MySQL passwords being presented to the web.

Alternate file layout

MediaWiki is designed to run in-place after being extracted from the distribution archive. This approach is convenient, but can result in reduced security or unnecessarily duplicated files.

You avoid duplicates in a mass installation or to keep sensitive files out of the web root for safety by manually relocating or consolidating various files.

Moving the main includes and skin files may require carefully picking and choosing and altering the `include_path` set in your `LocalSettings.php`. Experiment with this as desired.

If working to improve security, keep in mind that `WebStart.php` uses the current directory as a base. This means that only setting your `include_path` may not help to improve the security of your installation.

Move sensitive information

Consider moving the database password or other potentially sensitive data from `LocalSettings.php` to another file located outside of the web document root, and `include()` ing that file from `LocalSettings.php`. This can help to ensure that your database password will not be compromised if a web server configuration error disables PHP execution and reveals the file's source text.

Similarly, editing `LocalSettings.php` with some text editors will leave a backup file in the same directory with an altered file extension, causing the copy to be served as plain text if someone requests, for example, `LocalSettings.php~`. If you use such an editor, be sure to disable backup generation or move sensitive data outside the web root.

A MediaWiki debug logfile as it is used for debugging also contains sensitive data. Make sure to always disallow access for non authorised persons and the public as explained, delete remains of such logfiles when they are not needed, and comment or clear the logfile lines in your `LocalSettings.php`.

```
/**
 * The debug log file should never be publicly accessible if it is
 * used, as it
 * may contain private data. But it must be in a directory writeable
 * by the
 * PHP script running within your Web server.
 */
# $wgDebugLogFile = "c:/Program Files/Apache
Group/htdocs/mediawiki/debug.log" // Windows;
$wgDebugLogFile = "/tmp/${$wgSitename}-debug.log" // Linux;
```

Set DocumentRoot to /dev/null

A more secure option for the Apache Web Server is to set the `DocumentRoot` to an empty or non-existent directory, and then use `Alias` directives in the Apache configuration to expose only the scripts and directories that need to be web-accessible.

Loader scripts

A PHP-only solution that will work with any web server is to write a series of scripts that explicitly `chdir()` to a specific directory and then require one or more source files. For example:

```
<syntaxhighlight lang="php"><?php chdir('/path/to/wiki'); require('./index.php'); </syntaxhighlight>
```

User security

Anyone able to edit the user-interface system messages in the MediaWiki: namespace can introduce arbitrary HTML and JavaScript code into page output. This includes wiki users who have the *editinterface* permission, as well as anyone with direct write access to the text table in the database.

HTML is disabled on many system messages, particularly those displayed at the login screen, so the risk of password-snarfing JavaScript should be minimal. Malicious code could still attempt to exploit browser vulnerabilities (install spyware, etc), though, so, you should make sure that only trusted people can modify system messages.

Upload security

The main concern is: How do we prevent users from uploading malicious files?

File uploads are an optional feature of MediaWiki and are disabled by default. If you enable them, you also need to provide a directory in the web root which is writable by the web server user.

This has several implications for security:

- The directory may have to be world-writable, or else owned by the web server's limited user account. On a multiuser system it may be possible for other local users to slip malicious files into your upload directory (see multiuser notes above)
- While PHP's configuration sets a filesize limit on individual uploads, MediaWiki doesn't set any limit on total uploads. A malicious (or overzealous) visitor could fill up a disk partition by uploading a lot of files.
- Generated thumbnails and uploaded files held for overwrite confirmation may be kept in `images/thumb` and `images/tmp` without visible notice in the MediaWiki web interface. Keep an eye on their sizes as well.

The default configuration makes an attempt to limit the types of files which can be uploaded for safety:

- By default, file extensions `.png`, `.gif`, `.jpg`, and `.jpeg` are whitelisted (`$wgFileExtensions`).
- Various executable and script extensions are explicitly blacklisted (`$wgFileBlacklist`) even if you allow users to override the whitelist (`$wgStrictFileExtensions`).
- Several known image file extensions have their types verified using PHP's `getimagesize` ^[9] function.
- Uploaded files are checked to see if they could trip filetype detection bugs in Internet Explorer and Safari which might cause them to display as HTML.

In case these checks turn out to be insufficient, you can gain further protection by explicitly disabling server-side execution of PHP scripts (and any other scripting types you may have) in the uploads directory (by default, `images`).

For instance, an Apache `.conf` file fragment to do this if your MediaWiki instance is in `/Library/MediaWiki/web` might look something like:

```
<Directory "/Library/MediaWiki/web/images">
    # Ignore .htaccess files
    AllowOverride None

    # Serve HTML as plaintext, don't execute SHTML
    AddType text/plain .html .htm .shtml .php

    # Don't run arbitrary PHP code.
    php_admin_flag engine off

    # If you've other scripting languages, disable them too.
</Directory>
```

Your exact configuration may vary. In particular, the use of PHP's safe mode or open_basedir options may complicate handling of uploads.

External programs

- `/usr/bin/diff3` may be executed for edit conflict merging.
- If ImageMagick support for thumbnails or SVG images is enabled, `convert` may be run on uploaded files.
- If enabled, the `texvc` math extension will call `texvc` executable, which calls `latex`, `dvips`, and `convert` (which calls `gs`).

References

- [1] http://php.net/register_globals
- [2] <http://php.net/manual/en/filesystem.configuration.php#ini.allow-url-fopen>
- [3] <http://php.net/session>
- [4] <http://php.net/features.safe-mode>
- [5] <http://www.mediawiki.org/wiki/Manual:LocalSettings.php#Security>
- [6] <http://dev.mysql.com/doc/mysql/en/server-options.html>
- [7] <http://dev.mysql.com/doc/mysql/en/grant.html>
- [8] <http://dev.mysql.com/doc/mysql/en/create-database.html>
- [9] <http://php.net/manual/en/function.getimagesize.php>

Cross-site scripting

Cross-site scripting or XSS is, from the web app developer's perspective, another name for **arbitrary JavaScript injection**. It works like this:

- An attacker tricks an authenticated user into visiting a specially crafted URL, or a website which they control which can redirect them to the crafted URL.
- The URL points to your web app and includes Javascript in the query string. The web app, due to poor escaping, injects the arbitrary JavaScript into the page that gets shown to the user.
- The Javascript runs with full access to the user's cookies. It can modify the page in any way, and it can submit forms on behalf of the user. The risks are especially severe if the victim is an administrator with special privileges.

Example:

```
function getTableCell( $value ) {  
    global $wgRequest;  
    $class = $wgRequest->getVal( 'class' );  
    return "<td class='$class'>" . htmlspecialchars( $value ) . '</td>';  
}
```

The attacker sends the victim to a URL like:

```
http://example.com/wiki/SomePage?class='%20<script>hack();</script></td><td%20class=
```

Note that POST requests are also vulnerable, using offsite JavaScript.

To avoid this, the basic principles are:

- Validate your input
- Escape your output

You can skip validation, but you can never skip escaping. Escape everything.

It doesn't matter if the escaping is redundant with the validation, the performance cost is a small price to pay in exchange for a demonstrably secure web app. It doesn't matter if the input comes from a trusted source, escaping is necessary even then, because escaping gives you correctness as well as security.

Escape as close to the output as possible, so that the reviewer can easily verify that it was done. It helps you to verify your own code, as well.

Note that output encoding (escaping) is context sensitive. So be aware of the intended output context and encode appropriately (e.g. HTML entity, URL, Javascript, etc.)

The OWASP Abridged XSS Prevention Cheat Sheet ^[1] is a useful and up to date quick reference guide for mitigating XSS issues.

All this is true of any text-based interchange format. We concentrate on HTML because web apps tend to generate a lot of it, and because the security issues are particularly severe. Every text format should have a well-studied escaping function.

We also have some convenience functions in Xml.php which do HTML escaping for you.

Format	Escaping function	Notes
HTML	<code>htmlspecialchars()</code> ^[1]	Always use double quotes, single quotes aren't escaped by default
XML ID	<code>Sanitizer::escapeId()</code> ^[2]	For id attributes in HTML
Style	<code>Sanitizer::checkCss()</code> ^[3]	For style attributes in HTML
JavaScript	<code>FormatJson::encode()</code> ^[4] , <code>Xml::encodeJsVar()</code> ^[5]	
URL parameters	<code>wfArrayToCGI()</code> ^[6] , <code>urlencode()</code> ^[7]	
SQL	<code>\$db->addQuotes()</code> ^[8]	

MediaWiki also has some elegant building interfaces which implicitly escape things. For SQL using the 'key' => 'value' syntax of conditions implicitly escapes values. And the `Html::` and `Xml::` interface methods escape attributes, and depending on the method used may escape a text value as well.

Language:	English
-----------	---------

References

- [1] <http://php.net/manual/en/function.htmlspecialchars.php>
- [2] <http://svn.wikimedia.org/doc/classSanitizer.html#e091dfff62f13c9c1e0d2e503b0cab49>
- [3] <http://svn.wikimedia.org/doc/classSanitizer.html#ab64e762bd6ef22eec4a84f6d4be4c253>
- [4] <http://svn.wikimedia.org/doc/classFormatJson.html#4653b22cf81f08505850bf19290ee408>
- [5] <http://svn.wikimedia.org/doc/classXml.html#331bd4d134f54a51129e6d87a2ac52a8>
- [6] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#83e9f111af97c926c8634dd30d10b22e
- [7] <http://php.net/manual/en/function.urlencode.php>
- [8] <http://svn.wikimedia.org/doc/classDatabaseBase.html#6eb2da771487cd20ec6cacb26ce080f5>

Cross-site request forgery

Overview

Cross-site request forgery (or CSRF) is a type of attack ^[1] that uses web browser caching behavior to exploit vulnerabilities ^[2] in a web application's security.

CSRF attacks use authentication credentials cached in a victim's browser (such as a cookie or cached username and password) to authorize a malicious HTTP request. The malicious HTTP request can be sent in many ways. As long as the requests are processed by a web browser that has cached authentication credentials, a CSRF attack can be attempted. Examples of this include:

- links included in email messages that are opened in a browser by the victim;
- image tags embedded in HTML-formatted email (which are subsequently viewed in a web email client);
- forms embedded in a web page; or
- AJAX-style requests made via Javascript.

When successful, the attacks can allow the attacker to perform actions on a website with the same level of permissions as the victim.

For a broader overview of cross-site request forgeries, review Wikipedia's Cross-site request forgery page.

Example

The following code snippet helps demonstrate how a CSRF attack could occur. Assume that there are no other checks or protections in place to prevent an attack.

```
global $wgUser;  
if ( $wgUser->isAllowed('delete') && isset( $_POST['delete'] ) ) {  
    $this->deleteItem( $_POST['delete'] );  
}
```

The code allows a user to delete an item in wiki, as long as they have sufficient permissions and are authenticated.

If an authenticated user is tricked into visiting an external webpage under the control of an attacker, then the attacker can forge a request and send it to the wiki. The wiki would receive the request, check for authentication credentials, and then run the forged request as if it had actually been made by the victim.

Defending MediaWiki against CSRF attacks

We reduce the risk of CSRF attacks in MediaWiki using pseudo-random challenge tokens. These tokens are generated for each page that an authenticated user visits.

When submitting a form for a given page, the submission must include the token. Submissions that do not include the token will fail. It is difficult for attackers to guess or steal these tokens, making CSRF attacks more difficult.

Thanks to JavaScript's same origin policy, attackers cannot easily use Javascript to read the challenge token from the form.

Implementing Challenge Tokens

Challenge tokens should be added to all forms. You might want to look into migrating your code to `HTMLForm`, which has support for CSRF protection.

To add the tokens, use the `editToken()` method from the `$wgUser` global object.

The code will look like this:

```
function showForm() {
    ...
    $wgOut->addHTML( Xml::hidden( 'token', $wgUser->editToken() ) );
    ...
}

function submitForm() {
    ...
    if ( !$wgUser->matchEditToken( $wgRequest->getVal( 'token' ) ) ) {
        ... CSRF detected - stop the request right now ...
        return
    }
    // OK, continue submit
    ...
}
```

Every form which performs a write operation should be protected in this way.

See Also

- Manual:Edit_token
- Cross-Site Request Forgery^[3] at the The Open Web Application Security Project^[4]

References

- [1] <http://www.owasp.org/index.php/Attack>
- [2] <http://www.owasp.org/index.php/Vulnerability>
- [3] [http://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))
- [4] <http://www.owasp.org>

Register globals

Overview

Register globals ^[1] is a deprecated feature of PHP. The feature causes data passed to a PHP script via cookies or GET and POST requests to be made available as global variables in the script.

Register globals is convenient but extremely dangerous, often allowing an attacker to overwrite variables in a script simply by adding parameters to requests. While the feature has been disabled by default since PHP 4.2.0 ^[2] (which was released April 22, 2002), **hosting providers often enable the feature** to provide compatibility with old scripts.

Example

Here's a simple example of how register globals can present a security risk. The following PHP fragment is written with the expectation that `$file` will be set by a user submitting an HTML form containing a `<select name="file"><option>file1</option><option>file2</option><option>...</option></select>` tag.

```
<?php
# ...
# display the file specified by the <select name="file">...</select> form element
readfile( '/path/to/file/dir/' . $file );
# ...
```

Forcing the script to display a sensitive file containing passwords is simply a matter of making a GET request. For example, the following request could cause the script to display the contents of `LocalSettings.php`:

```
http://example.com/example.php?file=../../../../../../../../var/www/mediawiki/LocalSettings.php
```

While this would seem to rely on detailed knowledge of the server's file system, it is fast and easy to write scripts that explore many different paths very quickly.

Disabling Register Globals

When possible, ensure that register globals is disabled by setting `register_globals` ^[3] to `off` in your `php.ini` ^[4] file.

In cases where you cannot edit your `php.ini` file (or cannot disable the feature globally), you may still be able to disable the feature via your web server's configuration files. Note that `register_globals` cannot be set at runtime using PHP's `ini_set()` ^[5] function.

For the Apache Web Server, use the `php_flag` ^[6] directive in a `.htaccess` file to disable register globals on a per-directory tree basis.

```
php_flag register_globals off
```

Then use `phpinfo()` ^[2] to confirm that `ini_get()` ^[7] is set to `off`.

A detailed tutorial on the use of `.htaccess` files is outside of the scope of this documentation. For further information, read [Apache Tutorial: .htaccess files](#) ^[8].

Protecting MediaWiki from Register Globals

Do not use global variables in script paths

It is best to avoid using global variables in script paths. You'll be happier because your code reviewers will be happier and your code will have fewer vulnerabilities.

```
<?php
// Get common functions
require( dirname(__FILE__) . '/CommonFunctions.php' );
```

Make sure code is only executed in the right context

If for some reason it's absolutely necessary to use a global variable like this, you can protect it using some boilerplate code, present in many extensions:

```
<?php

if ( !defined( 'MEDIAWIKI' ) ) {
    echo "Not a valid entry point";
    exit( 1 );
}

require( "$IP/extensions/MyExtension/CommonFunctions.php" );
...
```

This ensures that the code can only be executed after MediaWiki is initialised. You can be sure that MediaWiki will set the \$IP variable when it initialises.

Sanitize custom global variables before use

Ensuring that code is executed in the correct context will only protect MediaWiki's default variables from register globals. Custom global variables will not be protected. In the following example, if register globals is enabled an attacker could still overwrite \$myExtPath.

```
<?php
if ( !defined( 'MEDIAWIKI' ) ) exit;

if ( !isset( $myExtPath ) ) {
    $myExtPath = "$IP/extensions/MyExtension";
}

require( "$myExtPath/CommonFunctions.php" );
...
```

If you must use a custom global variable (like \$myExtPath in the example above), ensure that it is initialized with a default value and is not used across include() ^[9] or require() ^[10].

We could make the above example safer by:

- ensuring that the script always sets \$myExtPath; and
- not using \$myExtPath in \$myExtPath/CommonFunctions.php unless we also set it in this file.

```
<?php
if ( ! defined( 'MEDIAWIKI' ) ) exit;
$myExtPath = "$IP/extensions/MyExtension";
```

```
require( "$myExtPath/CommonFunctions.php" );  
...
```

Configure extensions only after their setup file is included

Because MediaWiki uses global variables for its configuration namespace, this means that all extensions must be configured in LocalSettings.php **after** their setup file is included.

```
<?php  
  
$kittyCatName = 'Yoshi'; // set this here to avoid register_globals  
vulnerabilities  
  
function writeKittyName() {  
    global $wgOut;  
    global $kittyCatName; // definitely safe  
    $wgOut->addHTML( htmlspecialchars( $kittyCatName ) );  
}
```

In LocalSettings.php

```
require( "$IP/extensions/KittyCat/KittyCat.php" ); // sets default  
variables  
$kittyCatName = 'Puss'; // override the default
```

See Also

- Using Register Globals ^[11] - The php.net page on why you shouldn't use register globals.

References

- [1] <http://php.net/manual/en/security.globals.php>
- [2] <http://www.php.net/ChangeLog-4.php#4.2.0>
- [3] <http://www.php.net/manual/en/ini.core.php#ini.register-globals>
- [4] <http://php.net/manual/en/configuration.file.php>
- [5] http://php.net/ini_set
- [6] <http://www.php.net/manual/en/configuration.changes.php#configuration.changes.apache>
- [7] http://www.php.net/ini_get
- [8] <http://httpd.apache.org/docs/trunk/howto/htaccess.html>
- [9] <http://php.net/include>
- [10] <http://php.net/require>
- [11] <http://www.php.net/manual/en/security.globals.php>

SQL injection

Overview

SQL injection is a type of attack ^[1] that uses vulnerabilities ^[2] in an application's input validation or data typing for SQL queries.

When successful, the attack allows the attacker to inject data into an existing SQL query. The attacker may then be able to fetch private data, cause a denial of service or cause other unintended responses. In the worst case, the injected code would allow the attacker to gain full control of the system by exploiting multiple vulnerabilities in the database server, system utilities and operating system.

For an overview of SQL injection attacks, review Wikipedia's SQL Injection page.

Example

The following code snippet would allow an attacker to execute their own SQL commands (and is a syntax error in Oracle).

```
$limit = $wgRequest->getVal( 'limit' );
$res = $db->query( "SELECT * from kitties LIMIT $limit" );
```

The preferred way to run the above query would be:

```
$limit = $wgRequest->getVal( 'limit' );
$limit = intval( $limit ); // OPTIONAL validation
$res = $db->select( 'kitties', '*', false, __METHOD__,
    array( 'LIMIT' => $limit ) // REQUIRED automatic escaping
);
```

To exploit the vulnerability and fetch the email addresses of registered wiki users, the attacker would use a GET string of:

```
?limit=%201%20union%20select%20user_email%20from%20user;
```

SQL Injection and MediaWiki

MediaWiki has a custom SQL generation interface which has proven to be effective for eliminating SQL injection vulnerabilities. The SQL generation interface also provides DBMS abstraction and features such as table prefixes.

To keep MediaWiki safe from SQL injection:

- avoid using direct SQL queries at all costs;
- review Manual:Database access and use the functions provided in Database.php; and
- read the The Open Web Application Security Project ^[4]'s page on SQL injection (http://www.owasp.org/index.php/SQL_Injection).

Language:	English
-----------	---------

Database access

This article provides an overview of database access and general database issues in MediaWiki.

Database layout

For information about the MediaWiki database layout, such as a description of the tables and their contents, please see [Manual:Database layout and maintenance/tables.sql](#) ^[1].

Database Abstraction Layer

MediaWiki provides a database abstraction layer. Unless you are working on the abstraction layer, you should never directly call PHP's database functions (such as `mysql_query()` or `pg_send_query()`.)

The abstraction layer is accessed by using the `wfGetDB()` function. For more detailed documentation on `wfGetDB()`, see the entry on `wfGetDB()` ^[2] in the `GlobalFunctions.php` ^[3] file reference.

Typically, `wfGetDB()` is called with a single parameter, which can be `DB_SLAVE` (for read queries) or `DB_MASTER` (for write queries and read queries that need to have absolutely newest information) constant. The distinction between master and slave is important in a multi-database environment, such as Wikimedia. This function will return you a `Database` ^[4] object that you can use to access the database. See the [#Wrapper functions](#) section below for what you can do with this `Database` object.

To make a read query, something like this usually suffices:

```
$dbr = wfGetDB( DB_SLAVE );
$res = $dbr->select( /* ...see docs... */ );
foreach( $res as $row ) {
    ...
}
```

For a write query, use something like:

```
$dbw = wfGetDB( DB_MASTER );
$dbw->insert( /* ...see docs... */ );
```

We use the convention `$dbr` for read and `$dbw` for write to help you keep track of whether the database object is a slave (read-only) or a master (read/write). If you write to a slave, the world will explode. Or to be precise, a subsequent write query which succeeded on the master may fail when replicated to the slave due to a unique key collision. Replication on the slave will stop and it may take hours to repair the database and get it back online. Setting `read_only` in `my.cnf` on the slave will avoid this scenario, but given the dire consequences, we prefer to have as many checks as possible.

Wrapper functions

We provide a `query()` function for raw SQL, but the wrapper functions like `select()` and `insert()` are usually more convenient. They can take care of things like table prefixes and escaping for you under some circumstances. If you really need to make your own SQL, please read the documentation for `tableName()` and `addQuotes()`. You will need both of them.

Another important reason to use the high level methods rather than constructing your own queries is to ensure that your code will run properly regardless of the database type. Currently there is MySQL and reasonable support for SQLite and PostgreSQL, also somewhat limited for Oracle and DB2, but there could be other databases in the future such as MSSQL or Firebird.

In the following, the available wrapper functions are listed. For a detailed description of the parameters of the wrapper functions, please refer to class `DatabaseBase` ^[5]'s docs. Particularly see `DatabaseBase::select` ^[6] for an explanation of the `$table`, `$vars`, `$conds`, `$fname`, `$options`, and `$join_conds` parameters that are used by many of the other wrapper functions.

```
function select( $table, $vars, $conds = '', $fname =
'Database::select', $options = array() );
function selectRow( $table, $vars, $conds = '', $fname =
'Database::select', $options = array() );
function insert( $table, $a, $fname = 'Database::insert', $options =
array() );
function insertSelect( $destTable, $srcTable, $varMap, $conds, $fname =
'Database::insertSelect', $insertOptions = array(), $selectOptions =
array() );
function update( $table, $values, $conds, $fname = 'Database::update',
$options = array() );
function delete( $table, $conds, $fname = 'Database::delete' );
function deleteJoin( $delTable, $joinTable, $delVar, $joinVar, $conds,
$fname = 'Database::deleteJoin' );
function buildLike( /*...*/ );
```

Wrapper function: `select()`

The `select()` function provides the MediaWiki interface for a SELECT statement. The components of the SELECT statement are coded as parameters of the `select()` function. An example is

```
$dbr = wfGetDB( DB_SLAVE );
$res = $dbr->select(
    'category',                                // $table
    array( 'cat_title', 'cat_pages' ),         // $vars (columns
of the table)
    'cat_pages > 0',                           // $conds
    __METHOD__,                                // $fname =
'Database::select',
    array( 'ORDER BY' => 'cat_title ASC' )     // $options =
array()
);
```

This example corresponds to the query

```
SELECT cat_title, cat_pages FROM category WHERE cat_pages > 0 ORDER BY
cat_title ASC
```

Arguments are either single values (such as 'category' and 'cat_pages > 0') or arrays, if more than one value is passed for an argument position (such as array('cat_pages > 0', \$myNextCond)). If you pass in strings, you **must** manually use DatabaseBase::addQuotes() on your values as you construct the string, as the wrapper will not do this for you. The array construction for \$conds is somewhat limited; it can only do equality relationships (i.e. WHERE key = 'value').

Basic query optimization

MediaWiki developers who need to write DB queries should have some understanding of databases and the performance issues associated with them. Patches containing unacceptably slow features will not be accepted. Unindexed queries are generally not welcome in MediaWiki, except in special pages derived from QueryPage. It's a common pitfall for new developers to submit code containing SQL queries which examine huge numbers of rows. Remember that COUNT(*) is O(N), counting rows in a table is like counting beans in a bucket.

Replication

The largest installation of MediaWiki, Wikimedia, uses a large set of slave MySQL servers replicating writes made to a master MySQL server. It is important to understand the issues associated with this setup if you want to write code destined for Wikipedia.

It's often the case that the best algorithm to use for a given task depends on whether or not replication is in use. Due to our unabashed Wikipedia-centrism, we often just use the replication-friendly version, but if you like, you can use \$wgLoadBalancer->getServerCount() > 1 to check to see if replication is in use.

Lag

Lag primarily occurs when large write queries are sent to the master. Writes on the master are executed in parallel, but they are executed in serial when they are replicated to the slaves. The master writes the query to the binlog when the transaction is committed. The slaves poll the binlog and start executing the query as soon as it appears. They can service reads while they are performing a write query, but will not read anything more from the binlog and thus will perform no more writes. This means that if the write query runs for a long time, the slaves will lag behind the master for the time it takes for the write query to complete.

Lag can be exacerbated by high read load. MediaWiki's load balancer will stop sending reads to a slave when it is lagged by more than 30 seconds. If the load ratios are set incorrectly, or if there is too much load generally, this may lead to a slave permanently hovering around 30 seconds lag.

If all slaves are lagged by more than 30 seconds, MediaWiki will stop writing to the database. All edits and other write operations will be refused, with an error returned to the user. This gives the slaves a chance to catch up. Before we had this mechanism, the slaves would regularly lag by several minutes, making review of recent edits difficult.

In addition to this, MediaWiki attempts to ensure that the user sees events occurring on the wiki in chronological order. A few seconds of lag can be tolerated, as long as the user sees a consistent picture from subsequent requests. This is done by saving the master binlog position in the session, and then at the start of each request, waiting for the slave to catch up to that position before doing any reads from it. If this wait times out, reads are allowed anyway, but the request is considered to be in "lagged slave mode". Lagged slave mode can be checked by calling \$wgLoadBalancer->getLaggedSlaveMode(). The only practical consequence at present is a warning displayed in the page footer.

Lag avoidance

To avoid excessive lag, queries which write large numbers of rows should be split up, generally to write one row at a time. Multi-row INSERT ... SELECT queries are the worst offenders and should be avoided altogether. Instead do the select first and then the insert.

Working with lag

Despite our best efforts, it's not practical to guarantee a low-lag environment. Replication lag will usually be less than one second, but may occasionally be up to 30 seconds. For scalability, it's very important to keep load on the master low, so simply sending all your queries to the master is not the answer. So when you have a genuine need for up-to-date data, the following approach is advised:

1. Do a quick query to the master for a sequence number or timestamp
2. Run the full query on the slave and check if it matches the data you got from the master
3. If it doesn't, run the full query on the master

To avoid swamping the master every time the slaves lag, use of this approach should be kept to a minimum. In most cases you should just read from the slave and let the user deal with the delay.

Lock contention

Due to the high write rate on Wikipedia (and some other wikis), MediaWiki developers need to be very careful to structure their writes to avoid long-lasting locks. By default, MediaWiki opens a transaction at the first query, and commits it before the output is sent. Locks will be held from the time when the query is done until the commit. So you can reduce lock time by doing as much processing as possible before you do your write queries. Update operations which do not require database access can be delayed until after the commit by adding an object to `$wgPostCommitUpdateList`.

Often this approach is not good enough, and it becomes necessary to enclose small groups of queries in their own transaction. Use the following syntax:

```
$dbw = wfGetDB( DB_MASTER );
$dbw->begin();
/* Do queries */
$dbw->commit();
```

Use of locking reads (e.g. the FOR UPDATE clause) is not advised. They are poorly implemented in InnoDB and will cause regular deadlock errors. It's also surprisingly easy to cripple the wiki with lock contention. If you must use them, define a new flag for `$wgAntiLockFlags` which allows them to be turned off, because we'll almost certainly need to do so on the Wikimedia cluster.

Instead of locking reads, combine your existence checks into your write queries, by using an appropriate condition in the WHERE clause of an UPDATE, or by using unique indexes in combination with INSERT IGNORE. Then use the affected row count to see if the query succeeded.

Database schema

Don't forget about indexes when designing databases, things may work smoothly on your test wiki with a dozen of pages, but will bring a real wiki to a halt. See above for details.

For naming conventions, see [Manual:Coding conventions#Database](#).

SQLite compatibility

When writing MySQL table definitions or upgrade patches, it is important to remember that SQLite shares MySQL's schema, but that works only if definitions are written in a specific way:

- Primary keys must be declared within main table declaration, but normal keys should be added separately with CREATE INDEX:

Wrong	Right
<pre>CREATE TABLE /*_*/foo (foo_id INT NOT NULL AUTO_INCREMENT, foo_text VARCHAR(256), PRIMARY KEY(foo_id), KEY(foo_text));</pre>	<pre>CREATE TABLE /*_*/foo (foo_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, foo_text VARCHAR(256)); CREATE INDEX /*i*/foo_text ON foo (foo_text);</pre>


However, primary keys spanning over more than one field should be included in the main table definition:

```
CREATE TABLE /*_*/foo (  
  foo_id INT NOT NULL,  
  foo_text VARCHAR(256),  
  PRIMARY KEY(foo_id, foo_text)  
);  
  
CREATE INDEX /*i*/foo_text ON foo (foo_text);
```

- Don't add more than one column per statement:

Wrong	Right
<pre>ALTER TABLE /*_*/foo ADD foo_bar BLOB, ADD foo_baz INT;</pre>	<pre>ALTER TABLE /*_*/foo ADD foo_bar BLOB; ALTER TABLE /*_*/foo ADD foo_baz INT;</pre>

You can run basic compatibility checks with `php sqlite.php --check-syntax filename.sql`, or, if you need to test an update patch, `php sqlite.php --check-syntax tables.sql filename.sql`, assuming that you're in `$IP/maintenance/`.

**Databases**

Engines: MySQL – Oracle – PostgreSQL – SQLite – IBM DB2
Technical documentation: Schema (tables) – Access

Language:	English
-----------	---------

References

- [1] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/maintenance/tables.sql?view=markup>
- [2] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html#a44dcb6be36dfa8f9278fa98c066c82f4
- [3] http://svn.wikimedia.org/doc/GlobalFunctions_8php.html
- [4] <http://svn.wikimedia.org/doc/classDatabase.html>
- [5] <http://svn.wikimedia.org/doc/classDatabaseBase.html>
- [6] <http://svn.wikimedia.org/doc/classDatabaseBase.html#bc5f71fda50c08947bd8348d11980457>

Securing database passwords

LocalSettings.php

LocalSettings.php contains MySQL database user IDs and passwords. Keeping these passwords in LocalSettings.php is risky because php files can be served as plain text under several different conditions revealing your wiki admin account to the world. If you want to keep your admin account a secret remove them from LocalSettings.php

LocalSettings.php can be served as plain text if:

- Php is disabled on the server
- Php itself breaks
- You have cgi search.pl (a common cgi search script) anywhere in that domain. Description of exploit. ^[1]

Verify that apache can gain access to this file, and only administrators have access to this file when logged in.

Fix (Theory)

Check with your distro for what the apache user is (this varies, examples include "apache", "nobody", "httpd"). Then set the permissions for the folder you have installed mediawiki into (in this example "mediawikifolder") like so:

```
chown apache mediawikifolder
chgrp apache mediawikifolder
chmod o-rwx mediawikifolder (removes the access rights from other)
(probably repeat with g-rwx ... for LocalSettings.php )
make sure that u has r (or chmod 400 LocalSettings.php)
```

Note: The fix above only works if you are granted rights to change your wiki-folder owner and group to the apache's owner and group. If you do execute the above you get: Access denied. To prevent this, do:

```
chmod 710 "mediawikifolder"
```

Rights then required for LocalSettings.php are:

```
chmod 400 LocalSettings.php
```

or depending on your hosting

```
chmod 755 "mediawikifolder"
```

and

```
chmod 404 LocalSettings.php
```

In both cases there is no need for the executable bit.

Note: "mediawikifolder" is the folder where you put your mediawiki-installation (e.g. /var/www/mediawiki)

Example

There is no total security but having apache access the files as "other" is far away from being secure.

Default permissions set for mediawiki are root:root -rw-rw-r-- and lookup for directories.

I recommend setting ownership as above to *apache* (or as it must be named on your server). Now you either can set the group to *apache* also or to a group of very few people who shall be allowed to access the data. Remember: Don't act as root if possible! Instead act as a normal user who is in this group *staff*

Now make group *staff* or as you may want to name it. If it exists already you'll get an error.

```
kuser &                #on KDE with a gui, very simple
groupadd staff          #or on posix-terminal add per hand
addgroup putYourNameHere staff  #add him to staff
addgroup putNextNameHere staff  #add him to staff
addgroup root staff      #add him to staff
```

I recommend this setting (you can type these command one after one):

```
export mediawikifolder="/var/www/mw6" #put your setting here
export apacheID="www-data"            #put your setting here, see above for examples
export groupID="staff"                #example, name it as you like or as you've set it earlier
chown -R "$apacheID":"$groupID" $mediawikifolder
chmod -R 460 $mediawikifolder         #user can read, group can read and write others are not allowed
                                         #- be careful, directories come later
chmod -R 660 $mediawikifolder/images #we need write access for user(=apache) and group - you see,
                                         #noone else allowed
chmod 660 $mediawikifolder/config     # Allow write access for installation,
chmod -R u+X $mediawikifolder         #only user and staff can lookup (x on dirs)
chmod -R g+X $mediawikifolder
```

later set

```
chmod 060 $mediawikifolder/config      # Only staff
```

At this moment I can't see, why group *apache* has to be set. If you get problems with some modules or extensions you can do this:

```
export mediawikifolder="/var/www/mw6" #put your setting here
export groupID="apache"                #be sure to take right groupname that apache needs
chgroup -R $groupID $mediawikifolder
```

Keep Mysql Passwords Out Of Webroot

You should never put your mysql passwords in a text file that is within the web root. You can avoid doing so by doing this:

1. Make a directory outside your web root. For example, if your website is located at "/htdocs/www-wiki", then make a directory called "external_includes" outside of your webroot:
 1. mkdir /external_includes
2. Create a file in the directory you just made called something like "mysql_pw" and place a variable on a separate line for each of your mysql user name, password, hostname, and database name, each variable being set to the real values. For example, using vi as your editor:
 1. vi /external_includes/mysql_pw

2. i (vi command for insert)
3. Type the following lines using the real values of course in place of the bracketed "mysql_" fillers:
 1. \$db_host="[mysql_host]";
 2. \$db_name="[mysql_db_name]";
 3. \$db_user="[mysql_user]";
 4. \$db_password="[mysql_password]";

```
<?php
$db_host="[mysql_host] ";
$db_name="[mysql_db_name] ";
$db_user="[mysql_user] ";
$db_password="[mysql_password] ";
```

1. Take care to leave no whitespace (blank lines) after the text.
2. Save the file. In vi this is: [Escape Key]ZZ
3. Chmod and/or chown this file as explained in the previous example for LocalSettings.php so apache can read it. Usually something like:

```
chown apache /htdocs/external_includes/mysql_pw
chmod o-rw /htdocs/external_includes/mysql_pw
```

- Edit your LocalSettings.php file and add the following line in the beginning of the file:
`require_once("[FULL ABSOLUTE PATH TO mysql_pw]")` (in our example this would be: `require_once("/external_includes/mysql_pw");`)
- Now instead of your real password, user name, database name, and host name do this in LocalSettings.php:

```
$wgDBserver      = $db_host;
$wgDBname        = $db_name;
$wgDBuser        = $db_user;
$wgDBpassword    = $db_password;
```

This way if somebody is able to access and display LocalSettings.php, all they will see is the variables rather than the real password, username, etc. to your mysql database and the real file containing that information is off limits to the web server. You still need to make sure LocalSettings.php is only readonly to the apache user as described above.

NOTE. If you are doing these changes and do not have access to the users because you web server provider does not let you, then, from ftp the minimum rights you have to set for your "external_includes" are: "rwxr-xr-x" (755). For the file "mysql_pw" you will have to set "rwxr-xr--" (754), otherwise your wiki will not run. Still, your password is secure because the file with critical info is out of world access.

PHP breakage security problems

If your php breaks, it will serve LocalSettings.php as a regular file, giving the world your wiki database password!

Fix

(may break elsewhere!)

```
<IfModule !sapi_apache2.c>
  <Files ~ '\.php$'>
    Order allow,deny
    Deny from all
    Allow from none
  </Files>
  <Files ~ '\.phps'>
    Order deny,allow
    Allow from all
  </Files>
</IfModule>
```

Replace *sapi_apache2.c* with *mod_php4.c* for apache 1.3

Replace *sapi_apache2.c* with *mod_php5.c* for apache 2


Language:	English
-----------	---------

References

- [1] <http://www.securityspace.com/smysecure/catid.html?id=10627>

Extensions, SpecialPages, hooks & Co.

Extensions

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

What are extensions?



Extensions let you customize how MediaWiki looks and works.

Wiki users can browse through existing extensions or request a new extension. System administrators can install (or remove) extensions on the MediaWiki installations that they manage. Developers can write new extensions or improve existing extensions.

Search the extensions by keyword(s):

Depending on your goals you can use extensions to:

- extend the wiki markup used to write articles - see [Category:Parser function extensions](#) and [Category:Parser extensions](#) for examples.
- add new reporting and administrative capabilities - see [Category:Special page extensions](#) for examples.
- change the look and feel of MediaWiki - see [Gallery of user styles](#) and [Category:User interface extensions](#) for examples.
- enhance security via custom authentication mechanisms - see [Category:Authentication and Authorization Extensions](#) for examples.

Browsing extensions

You can browse [Category:Extensions](#) by category or the [Extension Matrix](#) to see the full range of extensions that have already been written. For information on installing these extensions or writing your own, see below.

Checking installed extensions

Only someone with administration access to the filesystem on a server can install extensions for MediaWiki, but anyone can check which extensions are active on an instance of MediaWiki by accessing the [Special:Version](#) article. For example, these extensions are active in the English Wikipedia.

Installing an extension

MediaWiki is ready to accept extensions just after installation is finished. To add an extension follow these steps:

1. Before you start

A few extensions require the installation of a patch. Many of them also provide instructions designed for installation using unix commands. You require shell access (SSH) to enter these commands listed on the extension help pages.

2. Download your extension.

Extension Distributor helps you to select and download most of the popular extensions.

Extensions are usually distributed as modular packages. They generally go in their own subdirectory of `$IP/extensions/`. A list of extensions documented on MediaWiki.org is available on the extension matrix, and a list of extensions stored in the Wikimedia SVN repository is located at `svn:trunk/extensions`^[1]. Some extensions are available as source code within this wiki^[2]. You may want to automate copying them.

Unofficial bundles of the extensions in the Wikimedia SVN repository can be found on the toolserver. These bundles are arbitrary snapshots, so keep in mind they might contain a broken version of the extension (just as if you load them from the developer's repository directly).

3. Install your extension.

Generally, at the end of the `LocalSettings.php` file (but above the PHP end-of-code delimiter, `"?>"`, if present), the following line should be added:

```
require_once( "$IP/extensions/extension_name/extension_name.php" );
```

This line forces the PHP interpreter to read the extension file, and thereby make it accessible to MediaWiki.

Some extensions can conflict with maintenance scripts, for example if they directly access `$_SERVER` (not recommended).

In this case they can be wrapped in the conditional so maintenance scripts can still run.


```
if ( !$wgCommandLineMode ) {
    require_once( "$IP/extensions/extension_name/extension_name.php" );
}
```

The maintenance script `importDump.php` will fail for any extension which requires customized namespaces which is *included* inside the conditional above such as `Extension:Semantic MediaWiki`, `Extension:Semantic Forms`.

Note: Ensure that required permissions are set for extensions!

Note: While this installation procedure is sufficient for most extensions, some require a different installation procedure. Check your extension's documentation for details.

Note: If you want to alter configuration variables in `LocalSettings.php`, you have to do this typically **after** including the extension. Otherwise defaults defined in the extension will overwrite your settings.

 **Caution:** While extension declaration *can* be placed in other places within the `LocalSettings.php` file, **never** place extensions before the `require_once("includes/DefaultSettings.php");` line. Doing so will blank the extension setup function arrays, causing no extensions to be installed, and probably will make your wiki inaccessible until you fix it!

Upgrading an extension

FIXME

Developing extensions


This complex topic is handled on under the developing extensions manual page.

References

[1] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/extensions/>

[2] <http://en.wikipedia.org/wiki/Special%3Aallpages?namespace=102>

Developing extensions

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Each extension consists of three parts:

1. setup,
2. execution, and
3. internationalization.

A minimal extension will consist of three files, one for each part:

MyExtension/*MyExtension.php*

Stores the setup instructions.

MyExtension/*MyExtension.body.php*

Stores the execution code for the extension. For complex extensions, requiring multiple PHP files, the implementation code may instead be placed in a subdirectory, *MyExtension/includes*. For an example, see the Semantic MediaWiki ^[1] extension.

MyExtension/*MyExtension.i18n.php*

stores internationalization information for the extension.

Note: Originally, extensions were single files, and you may still find some examples of this deprecated style.

When writing an extension, you should replace *MyExtension* above with the name of your extension. Files should be named in *UpperCamelCase*, which is the general file naming convention.^[2]

The three parts of an extension, setup, execution, and, internationalization as well as extension types and publishing your extension are described in the following sections of this page.



Setup

Your goal in writing the setup portion is to consolidate set up so that users installing your extension need do nothing more than include the setup file in their LocalSettings.php file, like this:

```
require_once( "$IP/extensions/myextension/myextension.php" );
```

If you want to make your extension user configurable, you need to define and document some configuration parameters and your users setup should look something like this:

```
require_once( "$IP/extensions/myextension/myextension.php" );
$wgMyExtensionConfigThis = 1;
$wgMyExtensionConfigThat = false;
```

To reach this simplicity, your setup file will need to accomplish a number of tasks (described in detail in the following sections):

- register any media handler, parser function, special page, custom XML tag, and variable used by your extension.
- define and/or validate any configuration variables you have defined for your extension.
- prepare the classes used by your extension for autoloading
- determine what parts of your setup should be done immediately and what needs to be deferred until the MediaWiki core has been initialized and configured
- define any additional hooks needed by your extension
- create or check any new database tables required by your extension.
- setup internationalization and localization for your extension


Registering features with MediaWiki

MediaWiki lists all the extensions that have been installed on its `Special:Version` page. For example, you can see all the extensions installed on this wiki at `Special:Version`. It is good form to make sure that your extension is also listed on this page. To do this, you will need to add an entry to `$wgExtensionCredits` for **each** media handler, parser function, special page, custom XML tag, and variable used by your extension. The entry will look something like this:

```
$wgExtensionCredits['validextensionclass'][] = array(
    'name' => 'Example',
    'author' => 'John Doe',
    'url' => 'http://www.mediawiki.org/wiki/User:JDoe',
    'description' => 'This extension is an example and performs no
discernible function',
    'version' => 1.5,
);
```

See `Manual:$wgExtensionCredits` for full details on what these fields do. Many of the fields are optional, but it's still good practise to fill them out.

In addition to the above registration, you must also "hook" your feature into MediaWiki. The above only sets up the `Special:Version` page. The way you do this depends on the type of your extension. For details, please see the documentation for each type of extension:


	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Making your extension user configurable

If you want your user to be able to configure your extension, you'll need to provide one or more configuration variables. It is a good idea to give those variables a unique name. They should also follow MediaWiki naming conventions (e.g. global variables should begin with \$wg).

For example, if your extension is named "Very silly extension that does nothing", you might want to name all your configuration variables to begin \$wgVsetdn or \$wgVSETDN. It doesn't really matter what you choose so long as *none* of the MediaWiki core begins its variables this way and you have done a reasonable job of checking to see that none of the published extensions begin their variables this way. Users won't take kindly to having to choose between your extension and some other extensions because you chose overlapping variable names.

It is also a good idea to include extensive documentation of any configuration variables in your installation notes.

 **Warning:** To avoid `register_globals` vulnerabilities, **ALWAYS** explicitly set all your extension's configuration variables in extension setup file. Constructs like `if (!isset($wgMyLeetOption)) $wgMyLeetOption = somevalue;` **do not** safeguard against `register_globals`!

Here is a rather complex example from the CategoryTree extension, showing the full range of possibilities here:

```
/**
 * Constants for use with the mode,
 * defining what should be shown in the tree
 */
define( 'CT_MODE_CATEGORIES', 0 );
define( 'CT_MODE_PAGES', 10 );
define( 'CT_MODE_ALL', 20 );
define( 'CT_MODE_PARENTS', 100 );

/**
 * Options:
 *
 * $wgCategoryTreeMaxChildren
 * - maximum number of children shown in a tree
 * node. Default is 200
 * $wgCategoryTreeAllowTag
 * - enable <categorytree> tag. Default is true.
 * $wgCategoryTreeDynamicTag
 * - loads the first level of the tree in a <categorytag>
 * dynamically. This way, the cache does not need to be
 * disabled. Default is false.
 * $wgCategoryTreeDisableCache
 * - disabled the parser cache for pages with a
 * <categorytree> tag. Default is true.
 * $wgCategoryTreeMaxDepth
 * - maximum value for depth argument; An array that maps
 * mode ; values to the maximum depth acceptable for the
 * depth option. Per default, the "categories" mode has a
 * max depth of 2, all other modes have a max depth of 1.
 * $wgCategoryTreeDefaultOptions
 * - default options for the <categorytree> tag.
```

```

* $wgCategoryTreeCategoryPageOptions
*      - options to apply on category pages.
* $wgCategoryTreeSpecialPageOptions
*      - options to apply on Special:CategoryTree.
*/

$wgCategoryTreeMaxChildren = 200;
$wgCategoryTreeAllowTag    = true;
$wgCategoryTreeDynamicTag  = false;
$wgCategoryTreeDisableCache = true;
$wgCategoryTreeMaxDepth    = array(
    CT_MODE_PAGES => 1,
    CT_MODE_ALL   => 1,
    CT_MODE_CATEGORIES => 2
);

# Default values for most options
$wgCategoryTreeDefaultOptions = array();
$wgCategoryTreeDefaultOptions['mode']           = null;
$wgCategoryTreeDefaultOptions['hideprefix']     = null;
$wgCategoryTreeDefaultOptions['showcount']      = false;
# false means "no filter"
$wgCategoryTreeDefaultOptions['namespaces']     = false;

# Options to be used for category pages
$wgCategoryTreeCategoryPageOptions = array();
$wgCategoryTreeCategoryPageOptions['mode']      = null;
$wgCategoryTreeCategoryPageOptions['showcount'] = true;

# Options to be used for Special:CategoryTree
$wgCategoryTreeSpecialPageOptions = array();
$wgCategoryTreeSpecialPageOptions['showcount']  = true;

```

Preparing classes for autoloading

If you choose to use classes to implement your extension, MediaWiki provides a simplified mechanism for helping php find the source file where your class is located. In most cases this should eliminate the need to write your own `__autoload($classname)` method.

To use MediaWiki's autoloading mechanism, you add entries to the variable `$wgAutoloadClasses`. The key of each entry is the class name; the value is the file that stores the definition of the class. For a simple one class extension, the class is usually given the same name as the extension, so your autoloading section might look like this (extension is named *MyExtension*):

```

$wgAutoloadClasses['MyExtension'] = dirname(__FILE__) .
'/MyExtension.body.php';

```

.

For complex extensions with multiple classes, your autoloading section might look like this:

```
$wgMyExtensionIncludes = dirname(__FILE__) . '/includes';

## Special page class
$wgAutoloadClasses['SpecialMyExtension']
    = $wgMyExtensionIncludes . '/SpecialMyExtension.php';

## Tag class
$wgAutoloadClasses['TagMyExtension']
    = $wgMyExtensionIncludes . '/TagMyExtension.php';
```

Deferring setup

LocalSettings.php runs early in the MediaWiki setup process and a lot of things are not fully configured at that point. This can cause problems for certain setup activities. To work around this problem, MediaWiki gives you a choice of when to run set up actions. You can either run them immediately by inserting the commands in your setup file -or- you can run them later, after MediaWiki has finished configuring its core software.

To defer setup actions, your setup file must contain two bits of code:

- the definition of a setup function
- the assignment of that function to the `$wgExtensionFunctions` array.

The PHP code should look something like this:

```
$wgExtensionFunctions[] = 'efFoobarSetup';

function efFoobarSetup() {
    #do stuff that needs to be done after setup
}
```

Adding database tables

If your extension needs to add its own database tables, use the `LoadExtensionSchemaUpdates` hook. See the manual page for more information on usage.

Execution

The technique for writing the implementation portion depends upon the part of MediaWiki system you wish to extend:

- **Wiki markup:** Extensions that extend wiki markup will typically contain code that defines and implements custom XML tags, parser functions and variables. You can click on any of the links in the preceding sentence to get full details on how to implement these features in your extension.
- **Reporting and administration:** Extensions that add reporting and administrative capabilities usually do so by adding special pages. For more information see [Manual:Special pages](#).
- **Article automation and integrity:** Extensions that improve the integration between MediaWiki and its backing database or check articles for integrity features, will typically add functions to one of the many hooks that affect the process of creating, editing, renaming, and deleting articles. For more information about these hooks and how to attach your code to them, please see [Manual:Hooks](#).
- **Look and feel:** Extensions that provide a new look and feel to mediaWiki are bundled into skins. For more information about how to write your own skins, see [Manual:Skin](#) and [Manual:Skinning](#).

- **Security:** Extensions that limit their use to certain users should integrate with MediaWiki's own permissions system. To learn more about that system, please see [Manual:Preventing access](#). Some extensions also let MediaWiki make use of external authentication mechanisms. For more information, please see [AuthPlugin](#). In addition, if your extension tries to limit readership of certain articles, please check out the gotchas discussed in [Security issues with authorization extensions](#).

See also the [Extensions FAQ](#), [Developer hub](#)

Internationalization

If you want your extension to be used on wikis that have a multi-lingual readership, you will need to add internationalization support to your extension. Fortunately this is relatively easy to do.

1. For any text string displayed to the user, define a message. MediaWiki supports parameterized messages and that feature should be used when a message is dependent on information generated at runtime. Assign each message a *lowercase* message id.
2. In your setup and implementation code, replace each literal use of the message with a call to `wfMsg($msgID, $param1, $param2, ...)`. Example: `wfMsg('addition', '1', '2', '3')`
3. Store the message definition in your internationalization file (*myextension.i18n.php*). This is normally done by setting up an array that maps language and message id to each string. Each message id should be lowercase and they may **not** contain spaces. A minimal file might look like this:

```
<?php
$messages = array();
$messages['en'] = array(
    'sillysentence' => 'This sentence says nothing',
    'answertoeverything' => 'Forty-two',
    'addition' => '$1 plus $2 equals $3', //sentence with params
);
$messages['fr'] = array(
    'sillysentence' => 'Une phrase absurde',
    'answertoeverything' => 'quarante-deux',
    'addition' => '$1 et $2 font $3', //phrase avec paramètres
);
```


4. In your setup routine, load the internalization file :

```
$wgExtensionMessagesFiles['myextension'] = dirname( __FILE__ ) .
'/myextension.i18n.php';
```

For more information, please see:

- [Internationalisation](#) - discusses the MediaWiki internationalization engine, in particular, there is a list of features that can be localized and some review of the MediaWiki source code classes involved in localization.
- [Localization checks](#) - discusses common problems with localized messages

Extension types

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Extensions can be categorized based on the programming techniques used to achieve their effect. Most complex extensions will use more than one of these techniques:

- **Subclassing:** MediaWiki expects certain kinds of extensions to be implemented as subclasses of a MediaWiki provided base class:
 - **Special pages** - Subclasses of the `SpecialPage` class are used to build pages whose content is dynamically generated using a combination of the current system state, user input parameters, and database queries. Both reports and data entry forms can be generated. They are used for both reporting and administration purposes.
 - **Skins** - Skins change the look and feel of MediaWiki by altering the code that outputs pages by subclassing the MediaWiki class `SkinTemplate`.
- **Hooks:** A technique for injecting custom php code at key points within MediaWiki processing. They are widely used by MediaWiki's parser, its localization engine, its extension management system, and its page maintenance system.
- **Tag-function associations** - XML style tags that are associated with a php function that outputs HTML code. You do not need to limit yourself to formatting the text inside the tags. You don't even need to display it. Many tag extensions use the text as parameters that guide the generation of HTML that embeds google objects, data entry forms, RSS feeds, excerpts from selected wiki articles.
- **Magic words:** A technique for mapping a variety of wiki text string to a single id that is associated with a function. Both variables and parser functions use this technique. All text mapped to that id will be replaced with the return value of the function. The mapping between the text strings and the id is stored in an array passed to each function attached to the `LanguageGetMagic` hook. The interpretation of the id is a somewhat complex process - see `Manual:Magic words` for more information.
 - **Variables** - Variables are something of a misnomer. They are bits of wikitext that look like templates but have no parameters and have been assigned hard-coded values. Standard wiki markup such as `{{ PAGENAME }}` or `{{ SITENAME }}` are examples of variables. They get their name from the source of their value: a php variable or something that could be assigned to a variable, e.g. a string, a number, an expression, or a function return value.
 - **Parser functions** - `{{functionname: argument 1 | argument 2 | argument 3...}}`. Similar to tag extensions, parser functions process arguments and returns a value. Unlike tag extensions, the result of parser functions is wikitext.
- **Ajax** - you can use AJAX in your extension to let your JavaScript code interact with your server side extension code, without the need to reload the page.

Support other core versions

You can visit the extension support portal to keep on top of changes in future versions of mediawiki and also add support for older versions that are still popular.

Publishing


To autocategorize and standardize the documentation of your existing extension, please see Template:Extension. To add your new extension to this Wiki: *Please replace "MyExtension" with your extension's name:*

Please also consult Writing an extension for deployment.

references

- [1] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/extensions/SemanticMediaWiki/>
- [2] <http://lists.wikimedia.org/pipermail/wikitech-l/2011-August/054839.html>

Parser functions

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Parser functions, added in MediaWiki 1.7, are a type of extension that integrate closely with the parser. The term "parser function" should not be confused with Extension:ParserFunctions, which is a collection of simple parser functions.



Description

Tag extensions are expected to take unprocessed text and return HTML; they have very little integration with the rest of the parser. For example, the output of a tag extension cannot be used as a template parameter. Also, expanding templates within a tag extension is possible, but it must be done manually — an error-prone process that changes from version to version.

The typical syntax for a parser function is:

Creating a parser function is slightly more complicated than creating a new tag because the function name must be a magic word — a keyword that supports aliases and localization.

Simple example

Below is an example of an extension that creates a parser function.

```
<?php

// Take credit for your work.
$wgExtensionCredits['parserhook'][] = array(

    // The full path and filename of the file. This allows MediaWiki
    // to display the Subversion revision number on Special:Version.
    'path' => __FILE__,

    // The name of the extension, which will appear on Special:Version.
```



```
'name' => "Example Parser Function",

// A description of the extension, which will appear on
Special:Version.
'description' => "A simple example parser function extension",

// The version of the extension, which will appear on
Special:Version.
// This can be a number or a string.
'version' => 1,

// Your name, which will appear on Special:Version.
'author' => "Me",

// The URL to a wiki page/web page with information about the
extension,
// which will appear on Special:Version.
'url' => "http://www.mediawiki.org/wiki/Manual:Parser_functions"
);

// Specify the function that will initialize the parser function.
$wgHooks['ParserFirstCallInit'][] =
'efExampleParserFunction_Initialize';

// Specify the function that will register the magic words for the
parser function.
$wgHooks['LanguageGetMagic'][] =
'efExampleParserFunction_RegisterMagicWords';

// Tell MediaWiki that the parser function exists.
function efExampleParserFunction_Initialize(&$parser) {

    // Create a function hook associating the "example" magic word with
the
    // efExampleParserFunction_Render() function.
    $parser->setFunctionHook('example',
'efExampleParserFunction_Render');

    // Return true so that MediaWiki continues to load extensions.
    return true;
}

// Tell MediaWiki which magic words can invoke the parser function.
function efExampleParserFunction_RegisterMagicWords(&$magicWords,
$langCode) {
```

```

    // Add the magic words.
    // If the first element of the array is 0, the magic word is case
insensitive.
    // If the first element of the array is 1, the magic word is case
sensitive.
    // The remaining elements in the array are "synonyms" for the magic
word.
    $magicWords['example'] = array(0, 'example');

    // Return true so that MediaWiki continues to load extensions.
    return true;
}

// Render the output of the parser function.
function efExampleParserFunction_Render($parser, $param1 = '', $param2
= '') {

    // The input parameters are wikitext with templates expanded.
    // The output should be wikitext too.
    $output = "param1 is $param1 and param2 is $param2";

    return $output;
}

```

With this extension enabled,

produces:

- param1 is hello and param2 is hi

Longer functions

For longer functions, you may want to split the hook functions out to a `_body.php` or `.hooks.php` file and make them static functions of a class. Then you can load the class with `$wgAutoloadClasses` and call the static functions in the hooks, e.g.:

Put this in your `MyExtension.php` file:

```

$wgAutoloadClasses['MyExtensionHooks'] = "$dir/MyExtension.hooks.php";
$wgHooks[' ... '][ ] = 'MyExtensionHooks::MyExtensionFunction';

```

Then put this in your `MyExtension.hooks.php` file:

```

class MyExtensionHooks {
    public static function MyExtensionFunction( ... ) { ... }
}

```

Caching

As with tag extensions, `$parser->disableCache()` may be used to disable the cache for dynamic extensions.

Parser interface

Controlling the parsing of output

To have the wikitext returned by your parser function be fully parsed (including expansion of templates), set the `noparse` option to false when returning:

```
return array( $output, 'noparse' => false );
```

It seems the default value for `noparse` changed from false to true, at least in some situations, sometime around version 1.12.

Conversely, to have your parser function return HTML that remains unparsed, rather than returning wikitext, use this:

```
return array( $output, 'noparse' => true, 'isHTML' => true );
```

However,

```
This is a test.
```

will produce something like this:

This is

param1 is hello and param2 is hi a test.

This happens due to a hardcoded "`\n\n`" that is prepended to the HTML output of parser functions. To avoid that and make sure the HTML code is rendered inline to the surrounding text, you can use this:

```
return $parser->insertStripItem( $output, $parser->mStripState );
```

Naming

By default, MW adds a hash character (number sign, "#") to the name of each parser function. To suppress that addition (and obtain a parser function with no "#" prefix), include the *SFH_NO_HASH* constant in the optional flags argument to `setFunctionHook`, as described below.

When choosing a name without a hash prefix, note that transclusion of a page with a name starting with that function name followed by a colon is no longer possible. In particular, avoid function names equal to a namespace name. In the case that interwiki transclusion [1] is enabled, also avoid function names equal to an interwiki prefix.

The `setFunctionHook` hook

For more details of the interface into the parser, see the documentation for `setFunctionHook` in `includes/Parser.php`. Here's a (possibly dated) copy of those comments:

function `setFunctionHook($id, $callback, $flags = 0)`

Parameters:

- string `$id` - The magic word ID
- mixed `$callback` - The callback function (and object) to use
- integer `$flags` - Optional, set it to the *SFH_NO_HASH* constant to call the function without "#".

Return value: The old callback function for this name, if any

Create a function, e.g., `{ {#sum: 1 | 2 | 3 } }`. The callback function should have the form:

```
function myParserFunction( $parser, $arg1, $arg2, $arg3 ) { ... }
```

The callback may either return the text result of the function, or an array with the text in element 0, and a number of flags in the other elements. The names of the flags are specified in the keys. Valid flags are:

found

The text returned is valid, stop processing the template. This is on by default.

nowiki

Wiki markup in the return value should be escaped

noparse

Unsafe HTML tags should not be stripped, etc.

noargs

Don't replace triple-brace arguments in the return value

isHTML


The returned text is HTML, armour it against wikitext transformation

	Languages:	English
---	-------------------	----------------

References

[1] <http://comments.gmane.org/gmane.org.wikimedia.mediawiki/16997>

Special pages, OutputPage, WebRequest, Database, User, Title.php, FAQ

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Special pages are pages that are created by the software on demand to perform a specific function. For example, a special page might show all pages that have one or more links to an external site or it might create a form providing user submitted feedback. Special pages are located in their own namespace (*Special:*) and are not editable directly like other pages. Developers can also create new special pages. These pages can be user-accessible and will generally show up in the list of all special pages at Special:SpecialPages. Some special pages are only accessible to users with certain permissions and accesses. Other special pages don't show up on the special page list at all and are only used by the wiki internally.



General Information

All of the ~75 built-in special pages that come with MediaWiki are called `SpecialSomename.php` and are located in the `includes/specials` directory. Special pages created by third party developers are generally stored in the `extensions` directory in their own file or as part of a larger extension. All special pages inherit from a class called `SpecialPage` which is defined in `includes/SpecialPage.php`. When a new special page is created, the user rights needed to access the page can be defined. These rights specify, among other things, whether

the page will show up on Special:Specialpages and whether the page is includable in other pages.

Special pages also have unique names that can be customized on a wiki. The general form is "Special:Pagename" where both "Special" and "Pagename" are customizable. The Special pseudo namespace can be translated in other languages. This translated namespace can be produced with the wikitext `{{ns:special}}`, on this wiki giving `""`. The **name** of the special page can also be redefined in a system message, for the site language, with the generic name of the special page as the ID.

A special page may or may not allow input. For example, Special:Export allows a user to define a specific page to export by calling Special:Export/Sun. If the special page allows complex input, additional parameters will be sent to the query string component of the the URL for processing, e.g. `http://www.mediawiki.org/w/index.php?title=Special:Recentchanges&days=3&limit=250`.

Notes:

- There are various ways to make special pages, but the one below is used by the bulk of official extensions, and adherence to this style is recommended. Also, be sure to include a credits block in the new special page for 'specialpage'. See `$wgExtensionCredits` for more details.
- After making a new special page, be sure to add it to Category:Special page extensions so other people can find it.
- This method is only valid for MediaWiki running on PHP5 and above. If you're using an earlier version of MediaWiki on a more-recent version of PHP, upgrade MediaWiki.
- Special pages cannot be included within frames unless you use `$wgOut->allowClickjacking()`;

Basic special page template

Most special page extensions require three files: a small setup file, which will be loaded every time MediaWiki starts, an internationalization file, and a file with the bulk of the code. All of them should be placed in a new directory inside the MediaWiki `extensions/` directory. MediaWiki coding conventions define the four files like this:

- `<ExtensionName>.php` - The setup file.
- `Special<SpecialPageName>.php` - The special page code.
- `.i18n.php` - The internationalization file.

Generally the special page should be named after the extension: so the Gadgets extension has the file `Gadgets.php` and `SpecialGadgets.php`. Obviously if your extension has more than one special page, you'll need more names.

In the example below, `<SpecialPageName>` is `MyExtension`. A working copy is available for download, see further down.

After creating the files listed below, add the following line to `LocalSettings.php`

- `require_once("$IP/extensions/MyExtension/MyExtension.php");`

The Setup File

The setup file (in this example, named `MyExtension.php`) looks like this:

```
<?php
# Alert the user that this is not a valid entry point to MediaWiki if
they try to access the special pages file directly.
if (!defined('MEDIAWIKI')) {
    echo <<<EOT

To install my extension, put the following line in LocalSettings.php:
require_once( "$IP/extensions/MyExtension/MyExtension.php" );
```

```

EOT;

    exit( 1 );
}

$wgExtensionCredits['specialpage'][] = array(
    'name' => 'MyExtension',
    'author' => 'My name',
    'url' => 'http://www.mediawiki.org/wiki/Extension:MyExtension',
    'description' => 'Default description message',
    'descriptionmsg' => 'myextension-desc',
    'version' => '0.0.0',
);

$dir = dirname(__FILE__) . '/';

$wgAutoloadClasses['SpecialMyExtension'] = $dir .
'SpecialMyExtension.php'; # Location of the SpecialMyExtension class
(Tell MediaWiki to load this file)
$wgExtensionMessagesFiles['MyExtension'] = $dir .
'MyExtension.i18n.php'; # Location of a messages file (Tell MediaWiki
to load this file)
$wgSpecialPages['MyExtension'] = 'SpecialMyExtension'; # Tell MediaWiki
about the new special page and its class name

```

This stub file registers several important things:

- The location of the SpecialMyExtension class
- The location of a messages file
- The new special page and its class name

You may also want to set a value for \$wgSpecialPageGroups so that your special page gets put under the right heading in Special:SpecialPages, e.g.:

```
$wgSpecialPageGroups['MyExtension'] = 'other';
```

The Special Page File

The body file (in this example, named SpecialMyExtension.php) will contain a subclass of SpecialPage. It will be loaded automatically when the special page is requested. In this example, the subclass MyExtension is called:

```

<?php
class SpecialMyExtension extends SpecialPage {
    function __construct() {
        parent::__construct( 'MyExtension' );
    }

    function execute( $par ) {
        global $wgRequest, $wgOut;

        $this->setHeaders();
    }
}

```

```

        # Get request data from, e.g.
        $param = $wgRequest->getText( 'param' );

        # Do stuff
        # ...
        $output = 'Hello world!';
        $wgOut->addWikiText( $output );
    }
}

```

`execute()` is the main function that is called when a special page is accessed. The function overloads the function `SpecialPage::execute()`. It passes a single parameter `$par`, the subpage component of the current title. For example, if someone follows a link to `Special:MyExtension/blah`, `$par` will contain "blah".

Wikitext and HTML output should normally be run via `$wgOut` -- do not use 'print' or 'echo' directly when working within the wiki's user interface.

However if you're using your special page as an entry point to custom XML or binary output, see [Taking over output](#) in your special page.

The Messages/Internationalization File

All special pages must specify a title to appear in the title and `<H1>` elements of the extension's page and on `Special:SpecialPages`. The extension specifies the title through a message. The structure of the message is a key-value pair. The ID, 'myextension', *must* be in all lowercase when specified in the key portion of the key-value pair, even if everywhere else it is `MyExtension`. The title, 'My Extension', can be anything, however it is convenient if the page title is the same as the ID. At its barest minimum, the message file (in this example, named `MyExtension.i18n.php`) should look like:

```

<?php
$specialPageAliases = array();
$messages = array();

$specialPageAliases['en'] = array(
    'MyExtension' => array( 'MyExtension', 'AnotherName' ),
);

$messages['en'] = array(
    'myextension' => 'My Extension',
);

$aliases =& $specialPageAliases; // for backwards compatibility with
MediaWiki 1.15 and earlier.

```

We have also specified the mapping between the name(s) of your special page and the underlying class. This allows the page title to be translated to another language. The page title can be customized into another language, the URL of the page would still be something like `.../Special:MyExtension`, even when the user language is not English. In this example, the special page `MyExtension` registers an *alias* so the page becomes accessible via `.../Special:AnotherName` eg. `.../Spezial:Meine_Erweiterung` in German, and so on.

The message file can contain a great deal more information. The following example adds a description of the extension. It also adds a text for browsers specifying a preference for German.

```
<?php
$messages = array();
$specialPageAliases = array();

/* *** English *** */
$specialPageAliases['en'] = array(
    'MyExtension' => array( 'MyExtension' ),
);
$messages['en'] = array(
    'myextension' => 'My Extension',
    'myextension-desc' => "Extension's description",
);

/* *** German (Deutsch) *** */
$specialPageAliases['de'] = array(
    'MyExtension' => array( 'MeineErweiterung', 'Meine Erweiterung' ),
);
$messages['de'] = array(
    'myextension' => 'Meine Erweiterung',
    'myextension-desc' => 'Beschreibung der Erweiterung',
);

$aliases =& $specialPageAliases; // For backwards compatibility with
MediaWiki 1.15 and earlier.
```

Note that IDs should not start with an uppercase letter, and that a space in the ID should be written in the code as an underscore. For the page header and linking, the usual rules for page names apply. If `$wgCapitalLinks` is true, a lowercase letter is converted to uppercase, and an underscore is displayed as a space. For example: instead of the above, we could have used `'my_extension' => 'My extension'`, assuming we consistently identified the extension as `my_extension` elsewhere.

Other Important Files

SpecialPage.php

Constructor

You can overload the constructor to initialize your own data, but the main reason you would want to do it is to change the behavior of the `SpecialPage` class itself. When you call the base class constructor from your child class, the following parameters are available:

```
function SpecialPage( $name = '', $restriction = '', $listed = true,
    $function = false, $file = 'default', $includable = false )
```

- *string* `$name` Name of the special page, as seen in links and URLs
- *string* `$restriction` User right required, e.g. "block" or "delete"

- *boolean* `$listed` Whether the page is listed in Special:Specialpages
- *string* `$function` A global function to call at run time if your subclass doesn't override `execute()`. By default it is constructed from `$name` as "wfSpecial\$name".
- *string* `$file` File which is included by `execute()`. It is also constructed from `$name` by default
- *boolean* `$includable` Whether the special page can be included from other pages using `{{Special:...}}`

SpecialPage->setHeaders()

This initialises the `OutputPage` object `$wgOut` with the name and description of your special page. It should always be called from your `execute()` method.

SpecialPage->including()

Some special pages can be included from within another page. For example, if you add `{{Special:RecentChanges}}` to the wikitext of a page, it will insert a listing of recent changes within the existing content of the page.

Including a special page from another web page is only possible if you declared the page to be includable in the constructor. You can do this by adding the following in the `__construct()` method after the parent class initialization:

```
$this->mIncludable = true;
```

You can also define your special page class as extending the `IncludableSpecialPage` class.

The `SpecialPage->including()` function returns a boolean value telling you what context the special page is being called from: `false` if it is a separate web page, and `true` if it is being included from within another web page. Usually you will want to strip down the presentation somewhat if the page is being included.

SpecialPage->execute()

This is the function which your child class should overload. It passes a single parameter, usually referred to cryptically as `$par`. This parameter is the subpage component of the current title. For example, if someone follows a link to `Special:MyExtension/blah`, `$par` will contain "blah".

OutputPage.php

The global variable `$wgOut` (of type `OutputPage`) is the variable you will use the most, because it is the way to send output to the browser (no, you don't use `echo` or `print`). If you want to use it somewhere, declare the variable global:

```
function randomFunction() {
    global $wgOut;
    $wgOut->addHTML('<b>This is not a pipe...</b>');
}
```

You can inspect the `OutputPage` class by viewing `includes/OutputPage.php` (indeed, all of these can be inspected), but there are a few methods you should definitely know about.

OutputPage->addHTML()

Essentially the quick and dirty substitute for `echo`. It takes your input and adds it to the buffer: no questions asked. In the below action, if `$action` contains user-data, it could easily have XSS, evil stuff, or the spawn of Satan injected in. You're better off using escaping (such as with the php function `htmlspecialchars`) or the XML builders class to build trusted output.

```
$wgOut->addHTML('<form action="'. $action. '" method="post">');
```

OutputPage->addWikiText()

For most output, you should be using this function. It's a bit of a black magic function: wikitext goes in, HTML comes out, and a whole lotta arcane code and demon summonings happen in between.

```
$wgOut->addWikiText("This is some ''lovely'' [[wikitext]] that will  
''get'' parsed nicely.");
```

What's worth noting is that the parser will view your chunks as cohesive wholes and paragraph accordingly. That is...

```
$wgOut->addWikiText('* Item 1');  
$wgOut->addWikiText('* Item 2');  
$wgOut->addWikiText('* Item 3');
```

Will output three lists with one item each, which probably wasn't intended.



Warning: If your special page is intended to be included in other pages, you should probably not use `addWikiText()` (or any other function that calls the parser). Due to a bug in MediaWiki^[1] (Bug 16129), an included special page will mess up any inclusion before it on the same including page, showing strings like `UNIQ10842e596cbb71da`.

workaround #1

As a workaround, you can have your extensions convert Wikitext to HTML using a separate Parser object and then use `addHTML()`. Example:

```
$wgOut->addHTML(sandboxParse("Here's some ''formatted'' text.));  
  
function sandboxParse($wikiText) {  
    global $wgTitle, $wgUser;  
    $myParser = new Parser();  
    $myParserOptions = new ParserOptions();  
    $myParserOptions->initialiseFromUser($wgUser);  
    $result = $myParser->parse($wikiText, $wgTitle, $myParserOptions);  
    return $result->getText();  
}
```

workaround #2

I tried the above, and found that the same problem now applied to any <tag>s in the transcluded text. This won't be a problem for a lot of extensions, but the extension I was writing was intended to show wikitext from another page as part of its functionality, so this was a problem.

The process for parsing a page which transcludes a special page seems to be this:

1. Replace {{Special:MyExtension}} with a UNIQ-QINU marker (because SpecialPage output is expected to be ready-to-output HTML)
2. Replace any <tag>s with QINU markers as above
3. Parse everything else from wikitext to HTML
4. Replace all QINU markers with their respective stored values, in a single pass

The process for parsing a page which transcludes a *non*-special page, though, is apparently like this:

1. Replace {{:Normal Article Name}} or {{Template Name}} with contents of transcluded page (because transcluded pages contain unparsed wikitext)
2. Replace any <tag>s with QINU markers as above
3. Parse everything else from wikitext to HTML
4. Replace all QINU markers with their respective stored values, in a single pass

The problem is apparently that in the earlier case, the parsing of the SpecialPage's wiki text is lacking the final QINU decoding step (why?), so all the QINU markers are left undecoded. (This may be a leftover from using the same syntax to invoke transclusion of a wikitext page, which is just pasted straight into the host page's wikitext contents and parsed, as is used to invoke transclusion of a SpecialPage, which must not be parsed at all. Wherever the code is that decides "wait, this is a special page -- replace it with a QINU", it should be doing the extra unstripGeneral before doing the QINU substitution.)

So I just did the following -- after this line:

```
$htOut = $wgParser->recursiveTagParse($iText);
```

...I added these lines (the second one is only because the function definition for the first one recommends it):

```
$htOut = $wgParser->mStripState->unstripGeneral($htOut);
$htOut = $wgParser->mStripState->unstripNoWiki($htOut);
```

Since I have now documented this, of course, I will now find a tragic flaw with it and feel really stupid... but as long as it seems to be working, I had to note it here. (It is also important to note the problem with work-around #1.) Also, I have only tested this with MediaWiki 1.10.1. The problem still exists under MW 1.14, but this solution may or may not work. --Woozle 18:26, 9 April 2009 (UTC)

workaround #3

The following function returns a parsed string without embedded parser comments

```
function efParse( $wikiText ) {
    global $wgOut;
    return $wgOut->parseInline( $wikiText );
}
```

Tested with MediaWiki 1.15.1 and PHP 5.2.12 --Wikinaut 12:31, 15 January 2010 (UTC)

wfMsg()

In most of the real special pages, you will rarely see `$wgOut->addWikitext()` without `wfMsg()` popping in. `wfMsg()` is a MediaWiki internationalization (i18n) function.

OutputPage->showErrorPage()

An error page is shown. The arguments `$title` and `$msg` specify keys into `wfMsg()`, not text. An example:

```
$wgOut->showErrorPage('error','badarticleerror');
```

- 'error' refers to the text *"Error"*.
- 'badarticleerror' refers to the text *"This action cannot be performed on this page."*

WebRequest.php

The `WebRequest` class is used to obtain information from the GET and POST arrays. Using this is recommended over directly accessing the superglobals, since the object does fun stuff like `magic_quotes` cleaning. The `WebRequest` object is accessible from extensions by including the global `$wgRequest` in the code.

WebRequest->getVal(\$key)

Returns a string that corresponds to the form input with the name `$key`.

WebRequest->get*()

Returns an int, bool, etc depending on the function called. For checkboxes for example, the function `getBool` is useful.

WebRequest->wasPosted()

Returns true if a form was posted.

Database.php

MediaWiki has a load of convenience functions and wrappers for interacting with the database. It also has an interesting load balancing scheme in place. It's recommended you use these wrappers. Check out `Database.php` for a complete listing of all the convenience functions, because these docs will only tell you about the non-obvious caveats. See `Manual:Database access`.

wfGetDB()

As this name suggests, this function gets you a reference of the database. There is no global that contains a database object.

When you call the function, you should pass it a parameter, the constant `DB_MASTER` or `DB_SLAVE`. Generally, you interact with the slave database when you're only performing read operations, and interact with the master when you're writing to the database. It's real easy to do, so do it, even if you only have one database.

User.php

The User class is used to represent users on the system. The global `$wgUser` represents the currently logged in user, and is usually what you will deal with when manipulating users.

User->isAllowed(\$right)

Returns true or false depending on whether the user is allowed to do \$right.

User->isBlocked()

Returns true if a user is blocked.

Title.php

Title represents the name of a page in the wiki. This is useful because MediaWiki does all sorts of fun escaping and special case logic to page names, so instead of rolling your own convert title to URL function, you create a Title object with your page name, and then use `escapeLocalURL()` to get a URL to that page.

FAQ

Setting an Extension Title

MediaWiki does not set the title of the extension, which is the developer's job. It will look for the name of the extension when `Special:Specialpages` is called or the special page is loaded (specifically right before the registered `wfSpecial*` function is called). Use `$wgOut` to title the extension like:

```
$wgOut->setPagetitle("your title");
```

The place where the extension can be found (as specified by what is passed into the `SpecialPage` constructor) is the key--**except** that it is not capitalized because of `getDescription()`, the internally used function that finds out the title (or, what they call description) of the special page, `strtolower` the name. "ThisIsACoolSpecialPage"'s key would be "thisisacoolspecialpage."

Theoretically, `getDescription` can be overloaded in order to avoid interacting with the message cache but, as the source code states: "Derived classes can override this, but usually it is easier to keep the default behavior. Messages can be added at run-time--see `MessageCache.php`". Furthermore, this prevents the MediaWiki namespace from overloading the message, as below.

Localizing the Extension Name

So you've just installed a shiny new MediaWiki extension and realize: "Oh no, my wiki is in French, but the page is showing up as English!" Most people wouldn't care, but it's actually a quite simple task to fix (as long as the developer used the method explained on this page). No noodling around in source code. Let's say the name of the page is `DirtyPages` and the name comes out to "List of Dirty Pages" but you want it to be (and excuse my poor French) "Liste de Pages Sales". Well, it's as simple as this:

1. Navigate to MediaWiki:DirtyPages, this page may not exist, but edit it anyway
2. Insert "Liste de Pages Sales" and save

And *voilà* (pardon the pun), the change is applied.

This is also useful for customizing the title for your wiki within your language: for instance, the developer called it "List of Dirty Pages" but you don't like that name, so you rename it "List of Pages needing Cleanup". Check out `Special:Allmessages` to learn more.

Also, if your extension has a large block of text that does change, like a warning, don't directly output the text. Instead, add it to the message cache and when the time comes to output the text in your code, do this:

```
$wgOut->addWikiText( wfMsg( 'dirtypageshelp' ) );
```

Then this message too can be customized at [MediaWiki:Dirtypageshelp](#).

See also [Help:System message](#).

Restricting the page to sysops or other users with special rights

Some special pages are omitted from the special page list unless the user has particular rights, such as being a sysop. To restrict your special page similarly, set the `restriction` parameter in the constructor. A typical sysop right is `editinterface`:

```
function __construct() {
    parent::__construct( 'MyExtension', 'editinterface' ); //
    restrict to sysops
}
```

Or you can create your own right in the setup file and assign it to sysops, e.g.:

```
$wgGroupPermissions['sysop']['myright'] = true;
$wgAvailableRights[] = 'myright';
```

and then call the constructor with your right:

```
function __construct() {
    parent::__construct( 'MyExtension', 'myright' );
}
```

Even if you restrict your page, your extension should check that the proper right is present, in case a user can somehow access it directly:

```
function execute( $par ) {
    // ...
    global $wgUser;
    if ( !$this->userCanExecute($wgUser) ) {
        $this->displayRestrictionError();
        return;
    }
    // ...
}
```

Disabling Special:UserLogin and Special:UserLogout pages

In LocalSettings.php you can use the SpecialPage_initList hook to *unset* unwanted built-in special pages. See "making a few SpecialPages restricted" ^[2] if you need *conditional* unsetting of special pages for example for certain user groups. The general message "You have requested an invalid special page." is shown if users try to access such unset special pages.

```
function disableSomeSpecialPages(&$list) {
    unset($list['Userlogout']);
    unset($list['Userlogin']);
    return true;
}
$wgHooks['SpecialPage_initList'][]='disableSomeSpecialPages';
```

Language:	English
-----------	---------

References

[1] <http://www.ehartwell.com/TechNotes/MediaWikiExtensionParserDebug.htm>
[2] <http://lists.wikimedia.org/pipermail/mediawiki-l/2009-June/031231.html>

Title.php

MediaWiki File: Title.php	
Location:	/includes/
Source code:	HEAD • 1.18beta1 1.18.0 • 1.17.1
Classes:	Title ^[1]

The MediaWiki software's `Title` class represents article titles, which are used for many purposes, including:

- as the human-readable text title of the article
- in the URL used to access the article
- the wikitext link to the article
- the key into the article database

The class is instantiated with one of these formats. Once instantiated, the title can retrieved in other formats or queried for its attributes. `Title` is intended to be an immutable "value" class, so there are no mutator functions.

To instantiate `Title`, call one of the static factory methods:

- `Title::newFromURL()`
- `Title::newFromDBKey()`
- `Title::newFromText()`

Once instantiated, the other non-static accessor methods can be used, such as `getText()`, `getDBKey()`, `getNamespace()`, etc.

Title structure

A title consists of an optional Interwiki prefix (such as "m:" for pages from mediawiki.org or "w:" for Wikipedia^[2] articles), followed by an optional namespace (such as "Manual:"), followed by the article name.

prefix: namespace: article name
optional optional required

Interwiki prefixes and namespaces

Interwiki prefixes and namespaces follow the same content rules:

- they must start with a letter
- they must end with a colon
- they may only contain digits, letters, the space character and the underscore character
- spaces and underscores may be used interchangeably
- they are case-insensitive

Interwiki prefixes and namespaces are only recognized if they are known to a given installation of MediaWiki, either by default or through configuration.

For example: On this wiki, "w:Name" is a link to the article "Name" on Wikipedia, because "w" is recognized as one of the allowable interwiki prefixes. The title "talk:Name" is a link to the article "name" in the "talk" namespace of the current wiki, because "talk" is a recognized namespace. Both may be present, and if so, the interwiki must come first, for example, "w:talk:name".

If a title begins with a colon as its first character, no prefixes are scanned for, and the colon is removed before the title is processed. Because of this rule, it is possible to have articles with colons in their names. "E. Coli 0157:H7" is a valid title, as is "2001: A Space Odyssey", because "E. Coli 0157" and "2001" are not valid interwikis or namespaces.

Article name

In the article name spaces and underscores are treated as equivalent and each is converted to the other in the appropriate context (underscore in URL and database keys, spaces in plain text). "Extended" characters in the 0x80..0xFF range are allowed in all places, and are valid characters. They are encoded in URLs. Extended characters are **not** urlencoded when used as text or database keys. Other characters may be ASCII letters, digits, hyphen, comma, period, apostrophe, parentheses and colon. No other ASCII characters are allowed, and will be deleted if found (they will probably cause a browser to misinterpret the URL).

Canonical forms

The canonical form of a title will always be returned by the object. In this form, the first (and only the first) character of the namespace and title will be uppercase; the rest of the namespace will be lowercase, while the title will be left as is.

The text form will use spaces, the URL and DBkey forms will use underscores. Interwiki prefixes are all lowercase. The namespace will use underscores when returned alone; it will use spaces only when attached to the text title.

`getArticleID()` needs some explanation: for "internal" articles, it should return the "page_id" field if the article exists, else it returns 0. For all external articles it returns 0. All of the IDs for all instances of Title created during a request are cached, so they can be looked up quickly while rendering wikitext with lots of internal links.

Example

To check and see if a given page already exists:

```
$titleObject = Title::newFromText( 'Talk:Your desired title here' );
if ( !$titleObject->exists() ) echo "There is no page with that name.";
```

Create a new Title from text, such as what one would find in a link. Decodes any HTML entities in the text. Spaces, prefixes, and an initial ':' indicating the main namespace are accepted. Note that if the page does not exist, this will not create it. For that, see Manual:Article.php.

References

[1] <http://svn.wikimedia.org/doc/classTitle.html>


[2] <http://wikipedia.org>

\$wgLegalTitleChars

Site customization: \$wgLegalTitleChars	
Override the default list of illegal characters in page titles.	
Introduced in version:	1.6.0 (r10960)
Removed in version:	<i>still in use</i>
Allowed values:	string of characters
Default value:	" %!\\"\$&'()*+,-./0-9:;=?@A-Z\\^_`a-z~\\x80-\\xFF" (+ was added in 1.8.0)

Other settings: Alphabetical | By Function

Details

 **Warning:** Don't change this unless you know what you're doing!

This is a regex character class (i.e. a list of characters in a format suitable for a regular expression) that you want MediaWiki to allow in page titles despite being in the list of illegal characters.

The list of illegal characters is as follows: #<>[]|{} , non-printable characters 0 through 31, and 'delete' character 127).

Problem characters

The following punctuation symbols may cause problems if enabled:

- `[]{}|#` - These are needed for link syntax, never enable them.
- `%` - Minor problems with path to query rewrite rules, see below. Included in the default allow list.
- `+` - Doesn't work with path to query rewrite rules, corrupted by apache. Included in the default allow list since MediaWiki 1.8.0. In some rare cases you may wish to remove `+` for compatibility with old links.
- `?` - Doesn't work with path to PATH_INFO rewrites. Included in the default allow list.

The last three of these punctuation problems can be avoided by using an alias, instead of a rewrite rule of either variety.

The problem with % is that when using a path to query rewrite rule, URLs are double-unescaped: once by Apache's path conversion code, and again by PHP. So %253F, for example, becomes "?". Our code does not double-escape to compensate for this, indeed double escaping would break if the double-escaped title was passed in the query string rather than the path. This is a minor security issue because articles can be created such that they are hard to view or edit.

Theoretically 0x80-0x9F of ISO 8859-1 should be disallowed, but this breaks interlanguage links and so they are included in the allowed list by default.

References

- revision 10960 ^[1] (09 September 2005)
- [Wikitech-1] importDump.php error, WikiRevision given a null title in import. ^[2]
- 1.8.3 change log ^[3]

References

- [1] <http://svn.wikimedia.org/viewvc/mediawiki?view=rev&revision=10960>
- [2] <http://comments.gmane.org/gmane.science.linguistics.wikipedia.technical/26833>
- [3] http://svn.wikimedia.org/viewvc/mediawiki/tags/REL1_8_3/phase3/RELEASE-NOTES?view=markup

Extending wiki markup

MediaWiki provides a standard text markup that can be easily customized. Both piecemeal and wholesale customizations are possible:

- **adding standard token types:** The standard approach to customized MediaWiki markup is to add new markup that looks like the built-in MediaWiki XML tags (*<tag>*), template (*{{...}}*), or link markup (*[[...]]*).
- **adding custom token types:** Some extensions define new token types. For example, Extension:ParserPhase2, adds several token types: *((%...%))*, *((@...@))*, and *((\$...\$))*.
- **fundamental changes to the parser:** A few extensions attempt to fundamentally change the parsing strategy so that markup from other sorts of wikis and content management can be used (must be used?) instead of the standard wiki markup.

Adding to the standard token types

- **Parser function extensions:** Parser function extensions extend parameterized template processing and typically look something like this: *{{#funcname ...}}*. Although any "template name" can be used, custom extensions always begin the function name with a #, as in the example above. Other parser function names are reserved for use by the MediaWiki core.

When *{{#funcname ...}}* is implemented as a parser function, it passes its template parameters to a PHP function instead of the usual template article. This function returns a string of wiki text that replaces the parameterized template. Parser functions are used to handle wiki text generation that involves logic that is too complex or confusing to write using normal template-writing techniques.

- **Variable extensions:** Variable extensions extend parameterless template processing.

Instead of the usual article transclusion, *{{XXX}}* is associated with a PHP function that returns a string of wiki text that replaces it. They are usually used to insert system information into wiki markup (e.g., the current time, the current page).

- **XML markup extensions:** XML markup extensions (also known as tag extensions) define custom XML-style tags in the wikitext:

```
<tagname parname="parvalue" ... parname="parvalue"> some text </tagname>
```

The text between the tags gets passed on to a PHP function which parses the contents of the tag and returns an HTML string that replaces the tag and text. The content inside the tags may be wiki markup, straight text, or text with formatting rules specific to the tag. It is up to the extension implementer. Please check documentation of individual extensions.


- **Link markup extensions:** Link markup extensions change the way MediaWiki interprets internal links, i.e., wiki markup of the form [[...]].
- **Extended syntax extensions:** Extended syntax extensions, mostly Magic Word extensions, add to the list of MediaWiki Magic Words, such as __NOTOC__. Usually, a specific PHP function interprets these words, and either replaces them with something, or sets some conditions for later processing during output generation, or both. Most usually, the replacement is the empty string; that is, the Magic Word is deleted, and nothing is shown in its place. Altered processing may involve addition of an extra piece of CSS, or suppression of user preference settings during page generation, and can be almost anything.

Adding new token types

To add new token types or to change the entire markup strategy, implementers need to add functions to one or more of the various parser and page output hooks:

- **Category:ParserBeforeStrip extensions** rely on the ParserBeforeStrip hook.
 - **Category:ParserAfterStrip extensions** rely on the ParserAfterStrip hook.
 - **Category:ParserBeforeInternalParse extensions** rely on the ParserBeforeInternalParse hook.
 - **Category:OutputPageBeforeHTML extensions** rely on the OutputPageBeforeHTML hook.
 - **Category:ParserBeforeTidy extensions** rely on the ParserBeforeTidy hook.
 - **Category:ParserAfterTidy extensions** rely on the ParserAfterTidy hook.
-

Magic words (Manual)

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Magic words are a technique for mapping a variety of wiki text strings to a single id that is associated with a function. Both variables and parser functions use this technique. All text mapped to that id will be replaced with the return value of the function. The mapping between the text strings and the id is stored in an array passed to each function attached to the `LanguageGetMagic` hook.



How magic words work

Whenever MediaWiki finds text between double braces (`{{XXX ...}}`) it must decide whether XXX is a variable, parser function, or template. To do so, it asks a series of questions:

1. **Does it have an associated magic word id?** As a first step in resolving markup of the form `{{XXX...}}`, MediaWiki attempts to translate XXX to a magic word id. The translation table is defined by functions attached to the `Manual:Hooks/LanguageGetMagic` hook.
 - If no magic word id is associated with XXX, XXX is presumed to be a template.
2. **Is it a variable?** If a magic word id *is* found, MediaWiki next checks to see if it has any parameters.
 - If no parameters are found, MediaWiki checks to see if the magic word id has been declared as a variable id. To check this, it retrieves the list of magic words serving by calling `MagicWord::getVariableIDs()`. This method gets its list of variable ids from a hard coded list of variable ids (see `Help:Variables`) and from a list of custom variable ids provided by all functions attached to the hook `MagicWordwgVariableIDs`.
 - If the magic word id has been classified as a variable, hooks MediaWiki calls the functions associated with the event name 'ParserGetVariableValueSwitch' until one is found that recognizes the magic word and can return its value.
3. **Is it a parser function?** If there are any parameters or if the magic word id is missing from the list of variable magic word ids, then MediaWiki assumes that the magic word is a parser function or template. If the magic word id is found in the list of parser functions declared via a call to `$wgParser->setFunctionHook($magicWordId, $renderingFunctionName)`, it is treated as a parser function and rendered using the function named `$renderingFunctionName`. Otherwise, it is presumed to be a template.

Defining magic words

For magic words to do their magic we must define two things:

- a mapping between wiki text and a magic word id
- a mapping between a magic word id and some php function that interprets the magic word.

Mapping wiki text to magic word ids

The `LanguageGetMagic` is used to associate each magic word id with a language-dependent array that describes all the text strings that mapped to the magic word id. For more information on defining hook functions, please see `Manual:Hooks`.

The first element of this array is a flag indicating whether or not the magic word is case sensitive. The remaining elements are a list of text that should be associated with the magic word id. If the case sensitive flag is 0, any case variant of the names in the array will match. If the case sensitive flag is 1, only exact case matches will be associated with the magic word id.

This association is created by a set of functions attached to `$wgHooks['LanguageGetMagic']`. Each of these functions expects two parameters: (`&$magicWords`, `$langCode`). `$magicWords` is an array to populate. `$langCode` is a w:ISO 639-2 letter language code. Implementers check the requested language, and add an element to `$magicWords`. The key is always the magic word id. The value is an array as described above. The return value of this method must always be `true` so that hooks that we registered after this one don't get ignored.

In the example below, a Spanish MediaWiki installation will associate the magic word id 'MAG_CUSTOM' with "#personalizado", "#custom", "#PERSONALIZADO", "#CUSTOM" and all other case variants. In an English MediaWiki only "#custom" in various case combinations will be mapped to 'MAG_CUSTOM':

```
$wgHooks['LanguageGetMagic'][] = 'wfAddCustomMagicWordLang';
function wfAddCustomMagicWordLang( &$magicWords, $langCode ) {

    #choose the text to id mapping based on language
    switch ( $langCode ) {
        case 'es':
            $magicWords['MAG_CUSTOM'] = array( 0, "#personalizado", "#custom"
        );
            break;
        default:
            $magicWords['MAG_CUSTOM'] = array( 0, "#custom" );
    }

    #tell MediaWiki to go on to the next hook
    return true;
}
```

Note: In MediaWiki versions prior to 1.8, `Manual:Hooks/LanguageGetMagic` did not pass a language parameter. To simulate this parameter, extensions backported to 1.7 or earlier can extract the language of `$wgContLang` on their own and proceed as normal and/or offer a language independent process for selecting the id-text mapping.

Associating a magic word id with a php function

The mechanism for associating magic word ids with rendering functions depends on whether the magic word will be used as a parser function or a variable. For more information, please see:

- [Manual:Parser functions](#)
- [Manual:Variables](#)

Registering magic words

In MediaWiki 1.8 and beyond there is no explicit requirement to register magic word ids. Registering the parser function or variables that use them is sufficient. For versions prior to 1.8, see below.

Magic words in MediaWiki versions before 1.8

MediaWiki versions prior to 1.8 differed in the following ways:

1. [Manual:Hooks/LanguageGetMagic](#) did not pass a language parameter. To simulate this parameter, extensions backported to 1.7 or earlier can extract the language of `$wgContLang` on their own and proceed as normal and/or offer a language independent process for selecting the id-text mapping.
2. Extensions that used magic words had to explicitly register their magic word ids using the hook `MagicWordMagicWords`. This method simply asked the implementer to supply the id of the magic word.

```
$wgHooks['MagicWordMagicWords'][] = 'wfAddCustomMagicWord';
function wfAddCustomMagicWord( &$magicWords ) {
    $magicWords[] = 'MAG_CUSTOM';  #magic word id
    return true;
}
```

Magic words (Help)

Magic words are strings of text that MediaWiki associates with a return value or function, such as time, site details, or page names. This page is about usage of standard magic words; for a technical reference, see [Manual:Magic words](#).

There are three general types of magic words:

- **Behavior switches:** these are uppercase words surrounded by double underscores, *e.g.* `__FOO__`
- **Variables:** these are uppercase words surrounded by double braces, *e.g.* `{{FOO}}`. As such, they look a lot like templates.
- **Parser functions:** these take parameters and are either of the form `{{foo: . . . }}` or `{{#foo: . . . }}`. See also [Help:Extension:ParserFunctions](#).

Page-dependent magic words will affect or return data about the *current* page (by default), even if the word is added through a transcluded template or included system message.

Behavior switches

A behavior switch controls the layout or behavior of the page and can often be used to specify desired omissions and inclusions in the content.

Word	Description	Versions
Table of contents		
<code>__NOTOC__</code>	Hides the table of contents (TOC).	
<code>__FORCETOC__</code>	Forces the table of content to appear at its normal position (above the first header).	
<code>__TOC__</code>	Places a table of contents at the word's current position (overriding <code>__NOTOC__</code>). If this is used multiple times, the table of contents will appear at the first word's position. It is possible to suppress the auto-generated section numbers, if the proper class exists locally at MediaWiki:Common.css , defined as <code>.tocnumber { display: none; }</code> .	
Editing		
<code>__NOEDITSECTION__</code>	Hides the section edit links beside headings.	
<code>__NEWSECTIONLINK__</code>	Adds a link ("+" by default) beside the "edit" tab for adding a new section on a non-talk page (see Adding a section to the end).	1.7+
<code>__NONEWSECTIONLINK__</code>	Removes the link beside the "edit" tab on pages in talk namespaces.	1.15+
Categories		
<code>__NOGALLERY__</code>	Used on a category page, replaces thumbnails in the category view with normal links.	1.7+
<code>__HIDDENCAT__</code>	Used on a category page, hides the category from the lists of categories in its members and parent categories (there is an option in the user preferences to show them).	1.13+
Language conversion		
<code>__NOCONTENTCONVERT__</code> <code>__NOCC__</code>	On wikis with language variants, don't perform any content language conversion (character and phase) in article display; for example, only show Chinese (zh) instead of variants like zh_cn, zh_tw, zh_sg, or zh_hk.	
<code>__NOTITLECONVERT__</code> <code>__NOTC__</code>	On wikis with language variants, don't perform language conversion on the title (all other content is converted).	
Other		
<code>__START__</code>	No effect.	

<code>__END__</code>	Explicitly marks the end of the article, to prevent MediaWiki from removing trailing whitespace. Removed in 19213.	1.1–1.8
<code>__INDEX__</code>	Tell search engines to index the page (overrides <code>\$wgArticleRobotPolicies</code> , but not <code>robots.txt</code>).	1.14+
<code>__NOINDEX__</code>	Tell search engines not to index the page (ie, do not list in search engines' results).	1.14+
<code>__STATICREDIRECT__</code>	On redirect pages, don't allow MediaWiki to automatically update the link when someone moves a page and checks "Update any redirects that point to the original title".	1.13+

Variables

Variables return information about the current page, wiki, or date. Their syntax is similar to templates. Variables marked as "[**expensive**]" are tracked by the software, and the number that can be included on a page is limited.

If a template name conflicts with a variable, the variable will be used (so to transclude the template `Template:PAGENAME` you would need to write `{{Template:PAGENAME}}`). In some cases, adding parameters will force the parser to invoke a template; for example, `{{CURRENTDAYNAME|x}}` transcludes `Template:CURRENTDAYNAME`, *not* the variable.

Date and time

The following variables return the current date and time in UTC.

Due to MediaWiki and browser caching, these variables frequently show when the page was *cached* rather than the current time.

The date and time magic words are formatted in the wiki content language. Since 1.19, they depend on the page content language.

Variable	Output	Description	Versions
Year			
<code>{{CURRENTYEAR}}</code>	2011	Year	
Month			
<code>{{CURRENTMONTH}}</code>	12	Month (zero-padded number)	
<code>{{CURRENTMONTHNAME}}</code>	December	Month (name)	
<code>{{CURRENTMONTHNAMEGEN}}</code>		Month (genitive form)	
<code>{{CURRENTMONTHABBREV}}</code>	Dec	Month (abbreviation)	1.5+
Day			
<code>{{CURRENTDAY}}</code>	7	Day of the month (unpadded number)	
<code>{{CURRENTDAY2}}</code>	07	Day of the month (zero-padded number)	1.6+
<code>{{CURRENTDOW}}</code>	3	Day of the week (unpadded number)	
<code>{{CURRENTDAYNAME}}</code>	Wednesday	Day of the week (name)	
Time			
<code>{{CURRENTTIME}}</code>	04:25	Time (24-hour HH:mm format)	
<code>{{CURRENTHOUR}}</code>		Hour (24-hour zero-padded number)	
Other			
<code>{{CURRENTWEEK}}</code>	49	Week (number)	
<code>{{CURRENTTIMESTAMP}}</code>	20111207042538	YYYYMMDDHHmmss timestamp	1.7+

The following variables do the same as the above, but using the site's server config or \$wgLocaltimezone.

- { {LOCALYEAR} }
- { {LOCALMONTH} }
- { {LOCALMONTHNAME} }
- { {LOCALMONTHNAMEGEN} }
- { {LOCALMONTHABBREV} }
- { {LOCALDAY} }
- { {LOCALDAY2} }
- { {LOCALDOW} }
- { {LOCALDAYNAME} }
- { {LOCALTIME} }
- { {LOCALHOUR} }
- { {LOCALWEEK} }
- { {LOCALTIMESTAMP} }

*For more thorough time formatting, you may want to install *Extension:ParserFunctions* to use the *#time* parser function*

Technical metadata

Note: Revision variables return data about the **latest edit to the current page**, even if viewing an older version of the page.

Variable	Output	Description	Versions
Site			
{ {SITENAME} }		The wiki's site name (\$wgSitename).	
{ {SERVER} }	http://en.wikipedia.org	domain URL (\$wgServer)	
{ {SERVERNAME} }	en.wikipedia.org	domain name (No longer dependent on \$wgServerName as of version 1.17)	
{ {DIRMARK} } { {DIRECTIONMARK} }		Outputs a unicode-directional mark that matches the wiki's default language's direction (<i>&lrm;</i> on left-to-right wikis, <i>&rlm;</i> on right-to-left wikis), useful in text with multi-directional text. Since 1.19, it depends on the page content language.	1.7+
{ {SCRIPTPATH} }		relative script path (\$wgScriptPath)	
{ {STYLEPATH} }		relative style path (\$wgStylePath)	1.16+
{ {CURRENTVERSION} }	1.7alpha	The wiki's MediaWiki version.	1.7+
{ {CONTENTLANGUAGE} } { {CONTENTLANG} }		The wiki's default interface language (\$wgLanguageCode)	1.7+
Latest revision to current page			
{ {REVISIONID} }	0	Unique revision ID	1.5+
{ {REVISIONDAY} }		Day edit was made (unpadded number)	1.8+
{ {REVISIONDAY2} }		Day edit was made (zero-padded number)	1.8+
{ {REVISIONMONTH} }		Month edit was made (zero-padded number as of 1.17+, unpadded number in prior versions)	1.8+
{ {REVISIONMONTH1} }		Month edit was made (unpadded number)	1.17+

{{REVISIONYEAR}}		Year edit was made	1.8+
{{REVISIONTIMESTAMP}}		Timestamp as of time of edit	1.8+
{{REVISIONUSER}}		The username of the user who made the most recent edit to the page, or the current user when previewing an edit	1.15+
{{PAGESIZE:page name}} {{PAGESIZE:page name R}}		[expensive] Returns the byte size of the specified page. Use " R" to get raw numbers.	1.13+
{{PROTECTIONLEVEL:action}}	protection level	Outputs the protection level (e.g. 'autoconfirmed', 'sysop') for a given action (e.g. 'edit', 'move') on the current page or an empty string if not protected.	1.15+
Affects page content			
{{DISPLAYTITLE:title}}		Format the current page's title header. The value must be equivalent to the default title: only capitalization changes and replacing spaces with underscores are allowed (this can be changed with \$wgRestrictDisplayTitle). It can be disabled or enabled by \$wgAllowDisplayTitle; disabled by default before 1.10+, enabled by default thereafter.	1.7+
{{DEFAULTSORT:sortkey}} {{DEFAULTSORTKEY:sortkey}} {{DEFAULTCATEGORYSORT:sortkey}} {{DEFAULTSORT:sortkey noerror}} {{DEFAULTSORT:sortkey noreplace}}		Used for categorizing pages, sets a default category sort key. For example if you put {{DEFAULTSORT:Smith, John}} at the end of John Smith, the page would be sorted under "S" by default in categories. It can take a second argument of <i>noerror</i> or <i>noreplace</i> to suppress error messages when multiple defaultsortkey's are used on one page or to make it do nothing if multiple defaultsortkey's are used.	1.10+ 1.19+ (for noerror and noreplace)

Statistics

Numbers returned by these variables normally contain separators (commas or spaces, depending on the local language), but can return raw numbers with the ":R" flag (for example, {{NUMBEROFPAGES}} → 0 and {{NUMBEROFPAGES:R}} → 0). Use "IR" for magic words that require a parameter like PAGESINCATEGORY (for example {{PAGESINCATEGORY:Help}} and {{PAGESINCATEGORY:Help|R}}). Also applicable to {{PAGESIZE:page name}} above.

The number magic words are formatted in the wiki content language. Since 1.19, it depends on the page content language.

Variable	Output	Description	Versions
Entire wiki			
{{NUMBEROFPAGES}}	0	Number of wiki pages.	1.7+
{{NUMBEROFARTICLES}}	0	Number of pages in content namespaces.	
{{NUMBEROFFILES}}	0	Number of uploaded files.	1.5+
{{NUMBEROFEDITS}}		Number of page edits.	1.10+
{{NUMBEROFVIEWS}}		Number of page views. Usually useless on a wiki using caching.	1.14+
{{NUMBEROFUSERS}}	0	Number of registered users.	1.7+
{{NUMBEROFADMINS}}		Number of users in the <i>sysop</i> group.	1.7+
{{NUMBEROFACTIVEUSERS}}		Number of active users, based on the criteria used in Special:Statistics.	1.15+

<code>{{PAGESINCATEGORY:categoryname}}</code> <code>{{PAGESINCAT:Help}}</code>		[expensive] Number of pages in the given category.	1.13+
<code>{{NUMBERINGROUP:groupname}}</code> <code>{{NUMINGROUP:groupname}}</code>	<code>{{NUMBERINGROUP:bureaucrat}}</code> used here)	Number of users in a specific group.	1.14+
<code>{{PAGESINNS:index}}</code> <code>{{PAGESINNAMESPACE:index}}</code>	<i>not enabled</i>	Number of pages in the given namespace (replace <i>index</i> with the relevant namespace index). For instance, <code>{{PAGESINNAMESPACE:14}}</code> will output the number of category pages. <code>{{PAGESINNS:0}}</code> differs from <code>{{NUMBEROFARTICLES}}</code> in that the former includes redirects and disambiguation pages. Disabled by default, enable with <code>\$wgAllowSlowParserFunctions</code> .	1.7+

Page names

Variable	Output	Description	Versions
<code>{{FULLPAGENAME}}</code>	Help:Magic words	Namespace and page title.	1.6+
<code>{{PAGENAME}}</code>	Magic words	Page title.	
<code>{{BASEPAGENAME}}</code>	Magic words	Page title excluding the current subpage and namespace ("Title/foo" on "Title/foo/bar"). For more complex splitting, use <code>{{#titleparts:}}</code> from ParserFunctions extension.	1.7+
<code>{{SUBPAGENAME}}</code>	Help:Magic words	The subpage title ("foo" on "Title/foo").	1.6+
<code>{{SUBJECTPAGENAME}}</code>	Help:Magic words	The namespace and title of the associated subject page.	1.7+
<code>{{TALKPAGENAME}}</code>	Help talk:Magic words	The namespace and title of the associated talk page.	1.7+

The `{{BASEPAGENAME}}` and `{{SUBPAGENAME}}` magic words only work in namespaces that have subpages enabled. See [Manual:\\$wgNamespacesWithSubpages](#) for information on enabling subpages.

The following are equivalents encoded for use in MediaWiki URLs (i.e. spaces replaced with underscores and some characters percent-encoded):

- `{{FULLPAGENAMEE}}`
- `{{PAGENAMEE}}`
- `{{BASEPAGENAMEE}}`
- `{{SUBPAGENAMEE}}`
- `{{SUBJECTPAGENAMEE}}`
- `{{TALKPAGENAMEE}}`

As of 1.15+, these can all take a parameter, allowing specification of the page to be operated on, instead of just the current page:

- `{{PAGENAME:Template:Main Page}}` → **Main Page**



Warning: Page titles containing certain characters, such as single quotes (') or asterisks *, may produce unexpected results when handled with these magic words, e.g. `{{PAGESINCATEGORY:{{PAGENAME}}}}`. See bugs 14779, 16474.

Note that `{{PAGENAME}}`, `{{PAGENAMEE}}` and `{{urlencode:}}` have distinct implementations. See [Manual:PAGENAMEE encoding](#) for details.

Namespaces

Variable	Output	Description	Versions
<code>{{NAMESPACE}}</code>	Help	Name of the page's namespace	
<code>{{SUBJECTSPACE}}</code> <code>{{ARTICLESPACE}}</code>	Help Help	Name of the associated content namespace	1.7+
<code>{{TALKSPACE}}</code>	Help talk	Name of the associated talk namespace	1.7+

The following are equivalents encoded for use in MediaWiki URLs (spaces replaced with underscores and some characters percent-encoded):

- `{{NAMESPACEE}}`
- `{{SUBJECTSPACEE}}`
- `{{TALKSPACEE}}`

As of 1.15+, these can take a full-page-name parameter and will return the requested namespace associated with that page, instead of with the current page:

- `{{NAMESPACE:Template:Main Page}}` → **Template**
- `{{SUBJECTSPACE:Template:Main Page}}` → **Template**
- `{{TALKSPACE:Template:Main Page}}` → **Template talk**

Parameter must not be a namespace name:

- `{{SUBJECTSPACE:Help talk}}` → "

Parser functions

Parser functions are very similar to variables, but take one or more parameters (technically, any magic word that takes a parameter is a parser function), and the name is sometimes prefixed with a hash to distinguish them from templates.

This page only describes parser functions that are integral to the MediaWiki software. Other parser functions may be added by MediaWiki extensions such as the [ParserFunctions](#) extension. For those see [Help:Extension:ParserFunctions](#).

URL data

Parser function	Input → Output	Description	Versions
<code>{{localurl:page name}}</code> <code>{{localurl:page name query_string}}</code>	<code>{{localurl:MediaWiki}}</code> → /wiki/MediaWiki <code>{{localurl:MediaWiki printable=yes}}</code> → /wiki/MediaWiki	The relative path to the title.	
<code>{{fullurl:page name}}</code> <code>{{fullurl:page name query_string}}</code> <code>{{fullurl:interwiki:remote page name query_string}}</code>	<code>{{fullurl:Category:Top level}}</code> → [1] <code>{{fullurl:Category:Top level action=edit}}</code> → [2]	A protocol-relative path to the title. This will also resolve Interwiki prefixes. Note: Unbracketed (plain) protocol-relative links are not automatically linked.	1.5+
<code>{{canonicalurl:page name}}</code> <code>{{canonicalurl:page name query_string}}</code> <code>{{canonicalurl:interwiki:remote page name query_string}}</code>	<code>{{canonicalurl:Category:Top level}}</code> → <code>{{canonicalurl:Category:Top level action=edit}}</code> →	The absolute path to the title, using the canonical url. This will also resolve Interwiki prefixes.	1.18+
<code>{{filepath:file name}}</code> <code>{{filepath:file name nowiki}}</code>	<code>{{filepath:Wiki.png}}</code> → <code>{{filepath:Wiki.png nowiki}}</code> → <code>{{filepath:Example.svg 300}}</code> →	The absolute URL to the full size or thumbnail (1.18+) of a media file.	1.12+ 1.18+
<code>{{urlencode:string}}</code> (or <code>{{urlencode:string QUERY}}</code>) <code>{{urlencode:string WIKI}}</code> <code>{{urlencode:string PATH}}</code>	WIKI to QUERY in 1.17; this may break templates that rely on this function.	The input encoded for use in URLs. Note that there is no urlencode function like there is in the obsolete Extension:StringFunctions.	1.7+ (or 1.17+) 1.17+ 1.17+
<code>{{anchorencode:string}}</code>	<code>{{anchorencode:x y z á é}}</code> → x_y_z_C3.A1_C3.A9	The input encoded for use in URL section anchors (after the '#' symbol in a URL).	1.8+

Namespaces


`{{ns:}}` returns the current localized name for the namespace with that index, canonical name, or local alias. Thus `{{ns:6}}`, `{{ns:File}}`, and `{{ns:Image}}` (an old name for the File namespace) all return "File". On a wiki where the content language was French, `{{ns:Fichier}}` would also be valid, but `{{ns:Datei}}` (the localisation of "File" into German) would not.


`{{nse:}}` is the equivalent encoded for MediaWiki URLs. It does the same, but it replaces spaces with underscores, making it usable in external links.

Content namespaces		Talk namespaces	
Usage	Output	Usage	Output
{{ns:-2}} or {{ns:Media}}	Media		
{{ns:-1}} or {{ns:Special}}	Special		
{{ns:0}} or {{ns:}}		{{ns:1}} or {{ns:Talk}}	Talk
{{ns:2}} or {{ns:User}}	User	{{ns:3}} or {{ns:User talk}}	User talk
{{ns:4}} or {{ns:Project}}	Project	{{ns:5}} or {{ns:Project talk}}	Project talk
{{ns:6}} or {{ns:File}} or {{ns:Image}}	File	{{ns:7}} or {{ns:File talk}} or {{ns:Image talk}}	File talk
{{ns:8}} or {{ns:MediaWiki}}	MediaWiki	{{ns:9}} or {{ns:MediaWiki talk}}	MediaWiki talk
{{ns:10}} or {{ns:Template}}	Template	{{ns:11}} or {{ns:Template talk}}	Template talk
{{ns:12}} or {{ns:Help}}	Help	{{ns:13}} or {{ns:Help talk}}	Help talk
{{ns:14}} or {{ns:Category}}	Category	{{ns:15}} or {{ns:Category talk}}	Category talk

Formatting

Usage	Input → Output	Description	Version
{{lc:string}}	{{lc:DATA CENTER}} → data center	The lowercase input.	1.5+
{{lcfirst:string}}	{{lcfirst:DATA center}} → dATA center	The input with the <u>very first</u> character lowercase.	1.5+
{{uc:string}}	{{uc:text transform}} → TEXT TRANSFORM	The uppercase input.	1.5+
{{ucfirst:string}}	{{ucfirst:text TRANSFORM}} → Text TRANSFORM	The input with the <u>very first</u> character uppercase.	1.5+

<pre>{{formatnum:unformatted num}}</pre> <pre>{{formatnum:formatted num R}}</pre>	<pre>{{formatnum:987654321.654321}}</pre> <p>→ 987654321.654321</p> <pre>{{formatnum:987,654,321.654321 R}}</pre> <p>→ 987,654,321.654321</p> <pre>{{formatnum:{{formatnum:987.654.321}}}}</pre> <p>→ 987 654 321 (e.g. with Italian locale)</p> <pre>{{formatnum:00001}}</pre> <p>→ 00001</p>	<p>The input with decimal and decimal group separators, and localized digit script, according to the wiki's default locale. The <code> R</code> parameter can be used to unformat a number, for use in mathematical situations. It doesn't always work if you try to unformat a number in non-English format on a wiki with non-English locale; if you need to format (according the wiki's locale) a number in unknown input format, try and use <code>formatnum</code> two times (but not if it can have a decimal group, or its separator will be eaten or the number won't be formatted).</p> <p> Warning: Leading zeroes are not removed, you can use <code>{{#expr:00001}}</code> instead if you have</p> <p>Extension:ParserFunctions installed</p>	<p>1.7+</p> <p>1.13+</p>
---	--	--	--------------------------

<code>format}}</code> <code>{{#formatdate:date format}}</code>	<pre> {{#dateformat:25 deC 2009 ymd}} → (your pref), 2009 DeC 25 (default) {{#formatdate:dec 25,2009 dmy}} → (your pref), 25 Dec 2009 (default) {{#dateformat:2009-12-25 mdy}} → (your pref), December 25, 2009 (default) {{#formatdate:2009 dec 25 ISO 8601}} → (your pref), 2009-12-25 (default) {{#dateformat:25 decEmber mdy}} → (your pref), DecEmber 25 (default) </pre> <p>Note: In the example above, "your pref" refers to your date preference on the current MediaWiki wiki only.</p>	<p>Formats an unlinked date based on user "Date format" preference, and adds metadata tagging it as a formatted date. For logged-out users and those who have not set a date format in their preferences, dates can be given a default: <code>mdy</code>, <code>dmy</code>, <code>ymd</code>, <code>ISO 8601</code> (all case sensitive). If only the month and day are given, only <code>mdy</code> and <code>dmy</code> are valid. If a format is not specified or is invalid, the input format is used as a default. If the supplied date is not recognized as a valid date (specifically, if it contains any metadata such as from a nested use of these or similar templates), it is rendered unchanged, and no (additional) metadata is generated.</p> <p> Warning: Although the ISO 8601 standard requires that dates be in the Gregorian calendar, the ISO parameter in this function will still format dates that fall outside the usual Gregorian range (e.g. dates prior to 1583). Also, the magic word cannot properly convert between negative years (used with ISO 8601) and years BC or years BCE (used in general writing).</p>	1.15+
<pre> {{padleft:xyz stringlength}} {{padleft:xyz strlen char}} {{padleft:xyz strlen string}} </pre>	<pre> {{padleft:xyz 5}} → 00xyz {{padleft:xyz 5 _}} → __xyz {{padleft:xyz 5 abc}} → abxyz {{padleft:xyz 2}} → xyz {{padleft: 1 xyz}} → x (first character of the string) </pre>	<p>Inserts a string of padding characters (character chosen in third parameter; default '0') of a specified length (second parameter) next to a chosen base character or variable (first parameter). The final digits or characters in the base replace the final characters in the padding; i.e.</p> <pre>{{padleft:44 3 0}}</pre> <p>produces 044. The padding string may be truncated if its length does not evenly divide the required number of characters.</p> <p>bug (fixed in r45734): multibyte characters are interpreted as two characters, which can skew width. These also cannot be used as padding characters.</p>	1.8+

<pre>{{padright:xyz stringlength}}</pre> <pre>{{padright:xyz strlen char}}</pre> <pre>{{padright:xyz strlen string}}</pre>	<pre>{{padright:xyz 5}} → xyz00</pre> <pre>{{padright:xyz 5 _}} → xyz__</pre> <pre>{{padright:xyz 5 abc}} → xyzab</pre> <pre>{{padright:xyz 2}} → xyz</pre> <pre>{{padright: 1 xyz}} → x</pre>	Identical to padleft, but adds padding characters to the right side.	
<pre>{{plural:2 is are}}</pre>	<pre>{{plural:0 is are}} →</pre> <pre>{{plural:1*1 is are}} →</pre> <pre>{{plural:21 mod 10 is are}} →</pre> <pre>{{plural:{{#expr:21 mod 10}} is are}} →</pre> <pre>{{plural:1 is are}} →</pre> <pre>{{plural:2 is are}} →</pre> <p>(for Polish):</p> <pre>{{plural:2 milion miliony milionów}} →</pre> <p>miliony</p>	Outputs the singular form (second parameter) if the first parameter is an expression equalling one; the plural form (third parameter) otherwise. Plural transformations are used for languages like Russian based on "count mod 10". You should not expect this to handle fractions (like 44.5) — see bug 28128.	
<pre>{{grammar:N noun}}</pre>		Outputs the correct inflected form of the given word described by the inflection code after the colon (language-dependent). Grammar transformations are used for inflected languages like Polish. See also Manual:\$wgGrammarForms.	1.7+

see also: Extension:StringFunctions

Miscellaneous

Usage	Output	Description	Version
<pre>{{int:message name}}</pre>	<pre>{{int:edit}} → "</pre> <p>(depends on user language, try: <i>fr</i>^[3] • <i>ja</i>^[4])</p>	Internationalizes (translates) the given interface (MediaWiki namespace) message into the user language. <i>Note that this can damage/confuse cache consistency in MediaWiki 1.17 and earlier, see bug 14404.</i>	
<pre>{{int:editsectionhint MediaWiki}}</pre>	<pre>{{int:editsectionhint MediaWiki}} → "</pre>	You may also use parameters with translations. Parameters are designated in messages with: \$1, \$2, \$3, etc. For example, here is the message for <i>editsectionhint</i> : Edit section: \$1 In this example, MediaWiki replaces \$1.	


<pre>{{#language:language code}} {{#language:ar}} {{#language:language code target language code}} {{#language:ar en}}</pre>	<p>language code</p> <p>ar</p> <p>language code</p> <p>Arabic</p>	<p>The full name of the language for the given language code: native name by default, name translated in target language if a target language code is specified. Codes are mostly in accordance with ISO 639.</p>	<p>1.7+ 1.18+ (translation)</p>
<pre>{{#special:special page name}} {{#special:userlogin}}</pre>		<p>The localized name for the given canonical Special: page.</p>	<p>1.9+</p>
<pre>{{#tag:tagname content parameter1=value1 parameter2=value2 }}</pre>	<p><i>(depends on parser tag)</i></p>	<p>and the attribute name.</p>	<p>1.12+</p>
<pre>{{gender:username return text if user is male return text if user is female return text if user hasn't defined their gender}}</pre>	<p><i>(depends on the named user's gender)</i></p>	<p>A switch for the gender set in Special:Preferences</p> <p>Note: If 3rd parameter is omitted and user hasn't defined his/her gender, then <i>text if user is male</i> is returned.</p>	<p>1.15+</p>

Language:	English
-----------	---------

References

- [1] http://en.wikipedia.org/wiki/Category%3Atop_level
- [2] http://en.wikipedia.org/wiki/Category%3Atop_level?action=edit
- [3] http://en.wikipedia.org/wiki/Help%3Amagic_words?uselang=fr#Miscellaneous
- [4] http://en.wikipedia.org/wiki/Help%3Amagic_words?uselang=ja#Miscellaneous

Hooks

	Tag Extensions	Parser Functions	Hooks	Special Pages	Skins	Magic Words
---	----------------	------------------	-------	---------------	-------	-------------

Hooks allow custom code to be executed when one of many defined events (like saving an article or a user logging in) occur. For example: The following code snippet, when added to LocalSettings.php, would call the function `MyExtensionHooks::articleSaveComplete` whenever the `ArticleSaveComplete` hook is ran (function arguments):



```
$wgHooks['ArticleSaveComplete'][] =
'MyExtensionHooks::articleSaveComplete';
```

MediaWiki provides many **hooks** that can be used to extend the functionality of the MediaWiki software. Assigning a function (known as an **event handler**) to a hook will cause that function to be called at the appropriate point in the main MediaWiki code, to perform whatever additional task(s) the developer thinks would be useful at that point. Each hook can have multiple handlers assigned to it, in which case it will call the functions in the order that they are assigned, with any modifications made by one function passed on to subsequent functions in the chain.

Hooks should be assigned at the *end* of LocalSettings.php or in your own extension file at the file scope (*not* in a `$wgExtensionFunctions` function or the `ParserFirstCallInit` hook). The easiest way to assign a function to a hook is:

```
$wgHooks['event'][] = 'function';
```

which adds an element to the array `$wgHooks`. You can also create new hooks in your own extension. Hooks created this way should be added to the Extension Hook Registry.

Background

Each hook is represented in the code by a call of function `wfRunHooks`, which is defined in file `Hooks.php`^[1]. The first argument of `wfRunHooks` is the name of the hook, the second is the array of arguments of the hook. Function `wfRunHooks` finds the tasks to be done from the array `$wgHooks`. It calls the PHP function `call_user_func_array`^[2] with arguments being the function to be called and its arguments.

See also the hook specification in SVN^[3].

In this example from the `doEdit` function in `Article.php`, `wfRunHooks` is used to call the `ArticleSaveComplete` hook using several arguments:

```
wfRunHooks( 'ArticleSaveComplete', array( &$this, &$user, $text,
$summary, $flags & EDIT_MINOR, &$watchthis, null,
    &$flags, $revision, &$status, $baseRevId, &$redirect) );
```

The core calls many hooks, but extensions can also call hooks.

Writing an event handler

An event handler is a function that is assigned to a hook, which will be run whenever the event represented by that hook occurs. It consists of:

- a function with some optional accompanying data, or
- an object with a method and some optional accompanying data.

Event handlers are registered by adding them to the global `$wgHooks` array for a given event. Hooks can be added from any point in the execution before the hook is called, but are most commonly added in `LocalSettings.php` or its included files. All the following are valid ways to define hooks, with the code that will be executed when 'EventName' happens:

Format	Syntax	Resulting function call.
Function, no data	<code>\$wgHooks['EventName'][] = 'someFunction';</code>	<code>someFunction(\$param1, \$param2);</code>
Function with data	<code>\$wgHooks['EventName'][] = array('someFunction', \$someData);</code>	<code>someFunction(\$someData, \$param1, \$param2);</code>
Function, no data (weird syntax, but OK)	<code>\$wgHooks['EventName'][] = array('someFunction');</code>	<code>someFunction(\$param1, \$param2);</code>
Object only	<code>\$wgHooks['EventName'][] = \$object;</code>	<code>\$object->onEventName(\$param1, \$param2);</code>
Object with method	<code>\$wgHooks['EventName'][] = array(\$object, 'someMethod');</code>	<code>\$object->someMethod(\$param1, \$param2);</code>
Object with method and data	<code>\$wgHooks['EventName'][] = array(\$object, 'someMethod', \$someData);</code>	<code>\$object->someMethod(\$someData, \$param1, \$param2);</code>
Object only (weird syntax, but OK)	<code>\$wgHooks['EventName'][] = array(\$object);</code>	<code>\$object->onEventName(\$param1, \$param2);</code>

When an event occurs, the function (or object method) will be called with the optional data provided as well as event-specific parameters. Note that when an object is the hook, and there's no specified method, the default method called is 'onEventName'. For different events this would be different: 'onArticleSave', 'onUserLogin', etc.

The extra data is useful if we want to use the same function or object for different purposes. For example:

```
$wgHooks['ArticleSaveComplete'][] = array('ircNotify', 'TimStarling');
$wgHooks['ArticleSaveComplete'][] = array('ircNotify', 'brion');
```

This code would result in `ircNotify` being run twice when an article is saved: once for 'TimStarling', and once for 'brion'.

Event handlers can return one of three possible values:

- `true`: the hook has operated successfully
- "some string": an error occurred; processing should stop and the error should be shown to the user
- `false`: the hook has successfully done the work necessary and the calling function should skip

Note: In many cases where an error message is expected, the hook will define a variable passed as a reference for extensions to store an error message and this is preferred over returning a string that will simply be displayed as an "internal error." In some cases hooks are only provided as a convenient way for extensions to read or modify data (e.g. `EditPageBeforeEditButtons`) and there is little difference between returning `true` and `false`. (Returning `false` causes any remaining functions assigned to the same hook to not run. However, hooks are added in the order extensions are run in `LocalSettings`, so relying on this for interactions between extensions can lead to inconsistent results.)

The last result would be for cases where the hook function replaces the main functionality. For example, if you wanted to authenticate users to a custom system (LDAP, another PHP program, whatever), you could do:

```
$wgHooks['UserLogin'][] = array('ldapLogin', $ldapServer);

$ldap['server'] = "ldaps://ldap.company.com/";
$ldap['port'] = 636;
$ldap['base'] = ",ou=Staff,dc=company,dc=com";

function ldapLogin( $username, $password ) {
    global $ldap;
    $auth_user = "uid=" . $username . $ldap['base'];
    wfSupressWarnings();
    if( $connect = ldap_connect( $ldap['server'], $ldap['port'] ) ){
        if( $bind = ldap_bind( $connect, $auth_user, $password ) ){
            ldap_close( $connect );
            return true;
        } else { // if bound to ldap
            echo 'Error on ldap_bind';
        }
    } else { // if connected to ldap
        echo 'Error on ldap_connect';
    }
    ldap_close( $connect );
    wfRestoreWarnings();
    return false;
}
```

Returning false makes less sense for events where the action is complete, and will normally be ignored.

Available hooks

This page contains a list of hooks that are made available by the MediaWiki software, and is known to be complete to version **1.8.2**. There is a lot of detail missing for the more recent hooks in particular, as their purpose/usage has not yet been documented by the developers. If you have any further information on any of these then please add it in the appropriate place.

In the tables, the first column gives the MediaWiki version that the hook was introduced in; use the link in the second column to find out more information about the hook and how to use it.

Note: For a complete list of hooks, use the category, which should be kept more up to date

Hooks grouped by function

Some of these hooks can be grouped into multiple functions.

Sections: Article Management - Edit Page - Page Rendering - User Interface - Special Pages - User Management - Logging - Skinning
Templates - API - Miscellaneous

Function	Version	Hook	Description
Article Management	1.4.0	ArticleDelete	Occurs whenever the software receives a request to delete an article
	1.4.0	ArticleDeleteComplete	Occurs after the delete article request has been processed
	1.9.1	ArticleUndelete	When one or more revisions of an article are restored
	1.12.0	ArticleRevisionUndeleted	Occurs after an article revision is restored
	1.8.0	ArticleFromTitle	Called to determine the class to handle the article rendering, based on title
	1.4.0	ArticleProtect	Occurs whenever the software receives a request to protect an article
	1.4.0	ArticleProtectComplete	Occurs after the protect article request has been processed
	1.6.0	ArticleInsertComplete	Occurs after a new article has been created
	1.11.0	RevisionInsertComplete	Called after a revision is inserted into the DB
	1.4.0	ArticleSave	Occurs whenever the software receives a request to save an article
	1.4.0	ArticleSaveComplete	Occurs after the save article request has been processed

	1.11.0	ArticleUpdateBeforeRedirect	Occurs after a page is updated (usually on save), before the user is redirected back to the page
	1.5.7	ArticleEditUpdateNewTalk	Allows an extension to prevent user notification when a new message is added to their talk page.
	1.14.0	ArticleEditUpdates	Called when edit updates (mainly link tracking) are made when an article has been changed.
	1.6.0	ArticleEditUpdatesDeleteFromRecentchanges	Occurs before saving to the database. If returning false old entries are not deleted from the recentchangeslist.
	1.8.0	RecentChange_save	Called after a "Recent Change" is committed to the DB
	1.6.0	SpecialMovepageAfterMove	Called after a page is moved.
	1.4.0	TitleMoveComplete	Occurs whenever a request to move an article is completed
	1.12.0	AbortMove	Allows to abort moving page from one title to another
	1.12.0	ArticleRollbackComplete	Occurs after an article rollback is completed
	1.12.0	ArticleMergeComplete	after merging to article using Special:Mergehistory
	1.6.0	ArticlePurge	Allows an extension to cancel a purge.
	1.13.0	ArticleRevisionVisibilitySet	called when changing visibility of one or more revisions of an article
Edit Page	1.6.0	AlternateEdit	Used to replace the entire edit page, altogether.
	1.6.0	EditFilter	Perform checks on an edit
	1.12.0	EditFilterMerged	Perform checks on an edit
	1.7.0	EditFormPreloadText	Called when edit page for a new article is shown. This lets you fill the text-box of a new page with initial wikitext.
	1.8.3	EditPage::attemptSave	Called before an article is saved, that is before insertNewArticle() is called
	1.6.0	EditPage::showEditForm:fields	Allows injection of form field into edit form.
	1.6.0	EditPage::showEditForm:initial	Allows injection of HTML into edit form
	1.13.0	EditPageBeforeConflictDiff	Allows modifying the EditPage object and output when there's an edit conflict.
	1.12.0	EditPageBeforeEditButtons	Used to modify the edit buttons on the edit form

Page Rendering	1.6.0	ArticlePageDataBefore	Executes before data is loaded for the article requested.
	1.6.0	ArticlePageDataAfter	Executes after loading the data of an article from the database.
	1.6.0	ArticleViewHeader	Called after an articleheader is shown.
	1.6.0	PageRenderingHash	Alter the parser cache option hash key.
	1.6.0	ArticleAfterFetchContent	Used to process raw wiki code after most of the other parser processing is complete.
	1.6.0	ParserClearState	Called at the end of Parser::clearState()
	1.5.0	ParserBeforeStrip	Used to process the raw wiki code before any internal processing is applied.
	1.5.0	ParserAfterStrip	(Deprecated) Before version 1.14.0, used to process raw wiki code after text surrounded by <code><nowiki></code> tags have been protected but before any other wiki text has been processed. In version 1.14.0 and later, runs immediately after ParserBeforeStrip.
	1.6.0	ParserBeforeInternalParse	Replaces the normal processing of stripped wiki text with custom processing. Used primarily to support alternatives (rather than additions) to the core MediaWiki markup syntax.
	1.6.0	ParserGetVariableValueVarCache	Use this to change the value of the variable cache or return false to not use it.
	1.6.0	ParserGetVariableValueTs	Use this to change the value of the time for the <code>{{LOCAL...}}</code> magic word.
	1.6.0	ParserGetVariableValueSwitch	Assigns a value to a user defined variable.
	1.10.0	InternalParseBeforeLinks	Used to process the expanded wiki code after <code><nowiki></code> , HTML-comments, and templates have been treated. Suitable for syntax extensions that want to customize the treatment of internal link syntax, i.e. <code>[[. . .]]</code> .
	1.13.0	LinkerMakeExternalLink	Called before the HTML for external links is returned. Used for modifying external link HTML.

1.13.0	LinkerMakeExternalImage	Called before external image HTML is returned. Used for modifying external image HTML.
1.11.0	EditSectionLink	(Deprecated) Override the return value of <code>Linker::editSectionLink()</code> . Called after creating [edit] link in header in <code>Linker::editSectionLink</code> but before HTML is displayed.
1.11.0	EditSectionLinkForOther	(Deprecated) Override the return value of <code>Linker::editSectionLinkForOther()</code> . Called after creating [edit] link in header in <code>Linker::editSectionLinkForOther</code> but before HTML is displayed.
1.14.0	EditSectionLink	(Deprecated) Override the HTML generated for section edit links.
1.5.0	ParserBeforeTidy	Used to process the nearly-rendered html code for the page (but before any html tidying occurs).
1.5.0	ParserAfterTidy	Used to add some final processing to the fully-rendered page output.
1.8.0	OutputPageParserOutput	Called after parse, before the HTML is added to the output.
1.6.0	OutputPageBeforeHTML	Called every time wikitext is added to the <code>OutputPage</code> , after it is parsed but before it is added. Called after the page has been rendered, but before the HTML is displayed.
1.4.3	CategoryPageView	Called before viewing a category page in <code>CategoryPage::view</code>
1.5.1	ArticleViewRedirect	Allows an extension to prevent the display of a "Redirected From" link on a redirect page.
1.10.0	IsFileCacheable	Allow an extension to disable file caching on pages.
1.10.1	BeforeParserFetchTemplateAndtitle	Allows an extension to specify a version of a page to get for inclusion in a template.
1.10.1	BeforeParserMakeImageLinkObj	Allows an extension to select a different version of an image to link to.
1.10.1	BeforeParserrenderImageGallery	Allows an extension to modify an image gallery before it is rendered.
1.17.0	ResourceLoaderRegisterModules	Allows registering modules with <code>ResourceLoader</code>

User Interface	1.5.4	AutoAuthenticate	Called to authenticate users on external/environmental means
	1.4.0	UserLoginComplete	Occurs after a user has successfully logged in
	1.4.0	UserLogout	Occurs when the software receives a request to log out
	1.4.0	UserLogoutComplete	Occurs after a user has successfully logged out
	1.6.0	userCan	To interrupt/advise the "user can do X to Y article" check
	1.4.0	WatchArticle	Occurs whenever the software receives a request to watch an article
	1.4.0	WatchArticleComplete	Occurs after the watch article request has been processed
	1.4.0	UnwatchArticle	Occurs whenever the software receives a request to unwatch an article
	1.4.0	UnwatchArticleComplete	Occurs after the unwatch article request has been processed
	1.6.0	MarkPatrolled	Called before an edit is marked patrolled
	1.6.0	MarkPatrolledComplete	Called after an edit is marked patrolled
	1.4.0	EmailUser	Occurs whenever the software receives a request to send an email from one user to another
	1.4.0	EmailUserComplete	Occurs after an email has been sent from one user to another
	1.6.0	SpecialMovepageAfterMove	Called after a page is moved.
	1.19.0	SpecialSearchCreateLink	Called when making the message to create a page or go to an existing page
	1.17.0	SpecialSearchGo	
	1.6.0	SpecialSearchNogomatch	
	1.19.0	SpecialSearchPowerBox	the equivalent of SpecialSearchProfileForm for the advanced form
	1.5.7	ArticleEditUpdateNewTalk	
	1.5.7	UserRetrieveNewTalks	
1.5.7	UserClearNewTalkNotification		
1.6.0	ArticlePurge		
File Upload	1.6.0	UploadVerification	Called when a file is uploaded, to allow extra file verification to take place (deprecated)
	1.17	UploadVerifyFile	Called when a file is uploaded, to allow extra file verification to take place (preferred)
	1.6.4	UploadComplete	Called when a file upload has completed.

Special pages	1.6.0	SpecialPageGetRedirect	
	1.13.0	SpecialListusersDefaultQuery	Called right before the end of UsersPager::getDefaultQuery()
	1.13.0	SpecialListusersFormatRow	Called right before the end of UsersPager::formatRow()
	1.13.0	SpecialListusersHeader	Called before closing the <fieldset> in UsersPager::getPageHeader()
	1.13.0	SpecialListusersHeaderForm	Called before adding the submit button in UsersPager::getPageHeader()
	1.13.0	SpecialListusersQueryInfo	Called right before the end of UsersPager::getQueryInfo()
	1.6.0	SpecialPageExecuteBeforeHeader	
	1.6.0	SpecialPageExecuteBeforePage	
	1.6.0	SpecialPageExecuteAfterPage	
	1.6.0	SpecialVersionExtensionTypes	
		SpecialPage_initList	Called after the Special Page list is populated
	1.9.0	UploadForm:initial	Called just before the upload form is generated
	1.9.0	UploadForm:BeforeProcessing	Called just before the file data (for example description) are processed, so extensions have a chance to manipulate them.
	1.14.0	UserRightsChangeableGroups	Called after the list of groups that can be manipulated via Special:UserRights is populated, but before it is returned.
User Management	1.5.0	AddNewAccount	Called after a user account is created
	1.5.8	AbortNewAccount	Can be used to cancel user account creation
	1.4.0	BlockIp	Occurs whenever the software receives a request to block an IP address or user
	1.4.0	BlockIpComplete	Occurs after the request to block an IP or user has been processed
	1.6.0	UserRights	Called after a user's group memberships are changed
	1.6.0	GetBlockedStatus	
Logging	1.6.0	LogPageActionText	
	1.5.0	LogPageLogHeader	
	1.5.0	LogPageLogName	
	1.5.0	LogPageValidTypes	

Skinning / Templates	1.7.0	BeforePageDisplay	Allows last minute changes to the output page, e.g. adding of CSS or Javascript by extensions.
	1.6.0	MonoBookTemplateToolboxEnd	Called by Monobook skin after toolbox links have been rendered (useful for adding more)
	1.7.0	PersonalUrls	(SkinTemplate.php) Called after the list of <i>personal URLs</i> (links at the top in Monobook) has been populated.
	1.11.0	SkinAfterBottomScripts	(Skin.php) At the end of Skin::bottomScripts()
	1.12.0	SkinSubPageSubtitle	(Skin.php) Called before the list of subpage links on top of a subpage is generated
	1.5.0	SkinTemplateContentActions	Called after the default tab list is populated (list is context dependent i.e. "normal" article or "special page").
	1.6.0	SkinTemplateTabs	Called after the skin's default tab list is populated.
	1.6.0	SkinTemplatePreventOtherActiveTabs	Called to enable/disable the inclusion of additional tabs to the skin.
	1.6.0	SkinTemplateSetupPageCss	
	1.6.0	SkinTemplateBuildContentActionUrlsAfterSpecialPage	
	1.6.0	SkinTemplateBuildNavUrlsNav_urlsAfterPermalink	Called after the <i>permalink</i> has been entered in navigation URL array.
	1.6.0	UserCreateForm	Allows for last minute updates to the UserCreateForm (SpecialUserLogin.php).
	1.6.0	UserLoginForm	Allows for last minute updates to the UserLoginForm (SpecialUserLogin.php).

API	1.13.0	APIEditBeforeSave	Called right before saving an edit submitted through <code>api.php?action=edit</code>
	1.13.0	APIQueryInfoTokens	Use this hook to add custom tokens to <code>prop=info</code>
	1.13.0	APIQueryRevisionsTokens	Use this hook to add custom tokens to <code>prop=revisions</code>
	1.14.0	APIQueryRecentChangesTokens	Use this hook to add custom tokens to <code>list=recentchanges</code>
	1.14.0	APIGetAllowedParams	Use this hook to modify a module's parameters
	1.14.0	APIGetParamDescription	Use this hook to modify a module's parameter descriptions
	1.14.0	APIAfterExecute	Use this hook to extend core API modules
	1.14.0	APIQueryAfterExecute	Use this hook to extend core API query modules
	1.14.0	APIQueryGeneratorAfterExecute	Use this hook to extend core API query modules
Miscellaneous	1.6.0	ArticleEditUpdatesDeleteFromRecentchanges	
	1.19.0	Collation::factory	Allows extensions to register new collation names, to be used with <code>\$wgCategoryCollation</code>
	1.19.0	GetDefaultSortkey	Allows to override what the default sortkey is, which is used to order pages in a category.
	1.6.0	GetInternalURL	Used to modify fully-qualified URLs (useful for squid cache purging)
	1.6.0	GetLocalURL	Used to modify local URLs as output into page links
	1.6.0	GetFullURL	Used to modify fully-qualified URLs used in redirects/export/offsite data
	1.16.0	ImgAuthBeforeStream	Executed before file is streamed to user, but only when using <code>img_auth</code>
	1.6.0	LanguageGetMagic	Used for parser function extensions
	1.14.0	LinkBegin	Used when generating internal and interwiki links in <code>Linker::link()</code>
	1.14.0	LinkEnd	Used when generating internal and interwiki links in <code>Linker::link()</code> , just before the function returns a value.
	1.6.0	MagicWordMagicWords	

	1.6.0	MagicWordwgVariableIDs	
	1.5.7	MessagesPreLoad	
	1.6.0	ParserTestParser	
	1.5.0	SpecialContributionsBeforeMainOutput	
	1.12.0	MediaWikiPerformAction	Override MediaWiki::performAction()
	1.4.0	UnknownAction	Used to add new query-string actions
	1.6.0	wgQueryPages	
	1.8.0	DisplayOldSubtitle	
	1.8.0	LoadAllMessages	
	1.8.0	RecentChange_save	Called after a "Recent Change" is committed to the DB
	1.8.0	UserToggles	Called before returning "user toggle names"

Alphabetical list of hooks

Version	Hook	Called From	Description
1.18.0	AbortAutoAccount	SpecialUserlogin.php	Allow to cancel automated local account creation, where normally authentication against an external auth plugin would be creating a local account.
1.13.0	AbortAutoblock	Block.php	Allow extension to cancel an autoblock
1.14.0	AbortDiffCache	DifferenceEngine.php	Can be used to cancel the caching of a diff
1.10.0	AbortLogin	SpecialUserlogin.php	Allows an extension like a captcha to abort the login process
1.12.0	AbortMove	Title.php, SpecialMovepage.php	Can be used to cancel page movement
1.5.8	AbortNewAccount	SpecialUserlogin.php	Can be used to cancel user account creation
1.18.0	ActionBeforeFormDisplay	Action.php	Before executing the HTMLForm object
1.18.0	ActionModifyFormFields	Action.php	Before creating an HTMLForm object for a page action
1.5.0	AddNewAccount	SpecialUserlogin.php	Called after a user account is created
1.17.0	AfterImportPage	ImportXMLReader.php	When a page import is completed
1.17.0	AfterUserMessage	User.php	After a user message has been left
1.9.1	AjaxAddScript	OutputPage.php	Called in output page just before the initialisation
1.6.0	AlternateEdit	EditPage.php	Occurs whenever action=edit is called
1.18.0	AlternateUserMailer	UserMailer.php	Called before mail is sent so that mail could be logged (or something else) instead of using PEAR or PHP's mail().
1.14.0	APIAfterExecute	ApiMain.php	Use this hook to extend core API modules
1.13.0	APIEditBeforeSave	ApiEditPage.php	Called right before saving an edit submitted through api.php?action=edit
1.14.0	APIGetAllowedParams	ApiBase.php	Use this hook to modify a module's parameters
1.14.0	APIGetParamDescription	ApiBase.php	Use this hook to modify a module's parameter descriptions

1.14.0	APIQueryAfterExecute	ApiQuery.php	Use this hook to extend core API query modules
1.14.0	APIQueryGeneratorAfterExecute	ApiQuery.php	Use this hook to extend core API query modules
1.13.0	APIQueryInfoTokens	ApiQueryInfo.php	Use this hook to add custom tokens to prop=info.
1.14.0	APIQueryRecentChangesTokens	ApiQueryRecentChanges.php	Use this hook to add custom tokens to list=recentchanges.
1.13.0	APIQueryRevisionsTokens	ApiQueryRevisions.php	Use this hook to add custom tokens to prop=revisions.
1.15.0	APIQueryUsersTokens	ApiQueryUsers.php	Use this hook to add custom tokens to list=users.
1.17.0	ApiRsdServiceApis	ApiRsd.php	Add or remove APIs from the RSD services list.
1.6.0	ArticleAfterFetchContent	Article.php	Used to process raw wiki code after most of the other parser processing is complete.
1.16.0	ArticleConfirmDelete	Article.php	Occurs before writing the confirmation form for article deletion.
1.17.0	ArticleContentOnDiff	DifferenceEngine.php	Before showing the article content below a diff.
1.4.0	ArticleDelete	Article.php	Occurs whenever the software receives a request to delete an article
1.4.0	ArticleDeleteComplete	Article.php	Occurs after the delete article request has been processed
1.5.7	ArticleEditUpdateNewTalk	Article.php	Allows an extension to prevent user notification when a new message is added to their talk page.
1.14.0	ArticleEditUpdates	Article.php	Called when edit updates (mainly link tracking) are made when an article has been changed.
1.6.0	ArticleEditUpdatesDeleteFromRecentchanges	Article.php	Occurs before saving to the database. If returning false old entries are not deleted from the recentchangeslist.
1.8.0	ArticleFromTitle	Wiki.php	Called to determine the class to handle the article rendering, based on title.
1.6.0	ArticleInsertComplete	Article.php	Called after an article is created
1.12.0	ArticleMergeComplete	SpecialMergeHistory.php	after merging to article using Special:Mergehistory
1.6.0	ArticlePageDataAfter	Article.php	after loading data of an article from the database
1.6.0	ArticlePageDataBefore	Article.php	before loading data of an article from the database
1.18.0	ArticlePrepareTextForEdit	Article.php	Called when preparing text to be saved.
1.4.0	ArticleProtect	Article.php	Occurs whenever the software receives a request to protect an article
1.4.0	ArticleProtectComplete	Article.php	Occurs after the protect article request has been processed
1.6.0	ArticlePurge	Article.php	Allows an extension to cancel a purge.
1.13.0	ArticleRevisionVisibilitySet	SpecialRevisiondelete.php	called when changing visibility of one or more revisions of an article
1.12.0	ArticleRevisionUndeleted	SpecialUndelete.php	Occurs after an article revision is restored
1.12.0	ArticleRollbackComplete	Article.php	Occurs after an article rollback is completed
1.4.0	ArticleSave	Article.php	Occurs whenever the software receives a request to save an article
1.4.0	ArticleSaveComplete	Article.php	Occurs after the save article request has been processed
1.9.1	ArticleUndelete	SpecialUndelete.php	When one or more revisions of an article are restored
1.11.0	ArticleUpdateBeforeRedirect	Article.php	Occurs after a page is updated (usually on save), before the user is redirected back to the page

1.19.0	ArticleViewCustom	Article.php	Allows to output the text of the article in a different format than wikitext.
1.18.0	ArticleViewFooter	Article.php	After showing the footer section of an ordinary page view.
1.6.0	ArticleViewHeader	Article.php	Occurs when header is shown
1.5.1	ArticleViewRedirect	Article.php	Allows an extension to prevent the display of a "Redirected From" link on a redirect page.
1.13.0	AuthPluginAutoCreate	SpecialUserlogin.php	Called when creating a local account for an user logged in from an external authentication method
1.9.1	AuthPluginSetup	Setup.php	Update or replace authentication plugin object (\$wgAuth)
1.5.4	AutoAuthenticate	StubObject.php	Called to authenticate users on external/environmental means.
1.12.0	AutopromoteCondition	Autopromote.php	check autopromote condition for user.
1.19.0	BacklinkCacheGetConditions	BacklinkCache.php	Allows to set conditions for query when links to certain title.
1.19.0	BacklinkCacheGetPrefix	BacklinkCache.php	Allows to set prefix for a specific link table.
1.7.0	BadImage	ImageFunctions.php	Before bad image list is evaluated
1.18.0	BaseTemplateToolbox	SkinTemplate.php	Called by BaseTemplate when building the toolbox array and returning it for the skin to output.
1.10.1	BeforeGalleryFindFile	ImageGallery.php	Allows extensions to specify a specific version of an image to display in a gallery.
1.16.0	BeforeInitialize	Wiki.php	Occurs before anything is initialized in MediaWiki::performRequestForTitle().
1.7.0	BeforePageDisplay	OutputPage.php SkinTemplate.php (< 1.12.0)	Allows last minute changes to the output page, e.g. adding of CSS or Javascript by extensions.
1.18.0	BeforeParserFetchFileAndTitle	Parser.php	Before an image is rendered by Parser
1.10.1	BeforeParserFetchTemplateAndtitle	Parser.php	Before a template is fetched by Parser
1.10.1	BeforeParserMakeImageLinkObj	Parser.php	Before an image is rendered by Parser
1.10.1	BeforeParserrenderImageGallery	Parser.php	Before an image gallery is rendered by Parser
1.12.0	BeforeWatchlist	SpecialWatchlist.php	Override watchlist display or add extra SQL clauses.
1.18.0	BitmapHandlerTransform	Bitmap.php	Before a file is transformed, gives extension the possibility to transform it themselves.
1.4.0	BlockIp	SpecialBlockip.php	Occurs whenever the software receives a request to block an IP address or user
1.4.0	BlockIpComplete	SpecialBlockip.php	Occurs after the request to block an IP or user has been processed
1.9.1	BookInformation	SpecialBooksources.php	Before information output on Special:Booksources
1.13.0	BrokenLink	Linker.php	Before the HTML is created for a broken (i.e. red) link
1.17.0	CanonicalNamespaces	Namespace.php	For extensions adding their own namespaces or altering the defaults.
1.4.3	CategoryPageView	CategoryPage.php	Called before viewing a category page in CategoryPage::view
1.12.0	ChangesListInsertArticleLink	ChangesList.php	Override or augment link to article in RC list.
1.19.0	Collation::factory	Collation.php	Allows extensions to register new collation names, to be used with \$wgCategoryCollation

1.16.0	ConfirmEmailComplete	User.php	Called after a user's email has been confirmed successfully.
1.13.0	ContribsPager::getQueryInfo	SpecialContributions.php	Before the contributions query is about to run
1.13.0	ContributionsLineEnding	SpecialContributions.php	Called before a contributions HTML line is finished
1.11.0	ContributionsToolLinks	SpecialContributions.php	Change tool links above Special:Contributions
1.9.1	CustomEditor	Wiki.php	When invoking the page editor. Return true to allow the normal editor to be used, or false if implementing a custom editor, e.g. for a special namespace, etc.
1.16.0	DatabaseOraclePostInit	DatabaseOracle.php	Called after initialising an Oracle database connection
1.7.0	DiffViewHeader	DifferenceEngine.php	Called before diff display
1.8.0	DisplayOldSubtitle	Article.php	Allows extensions to modify the displaying of links to other revisions when browsing through revisions.
1.14.0	DoEditSectionLink	Linker.php	Override the HTML generated for section edit links.
1.6.0	EditFilter	EditPage.php	Perform checks on an edit
1.12.0	EditFilterMerged	EditPage.php	Perform checks on an edit
1.7.0	EditFormPreloadText	EditPage.php	Called when edit page for a new article is shown. This lets you fill the text-box of a new page with initial wikitext.
1.16.0	EditFormInitialText	EditPage.php	Called when edit page for a existing article is shown. This lets you modify the content.
1.8.3	EditPage::attemptSave	EditPage.php	Called before an article is saved, that is before insertNewArticle() is called
1.16.0	EditPage::importFormData	EditPage.php	Allow extensions to read additional data posted in the form
1.6.0	EditPage::showEditForm:fields	EditPage.php	Allows injection of form field into edit form.
1.6.0	EditPage::showEditForm:initial	EditPage.php	before showing the edit form
1.13.0	EditPageBeforeConflictDiff	EditPage.php	Allows modifying the EditPage object and output when there's an edit conflict.
1.12.0	EditPageBeforeEditButtons	EditPage.php	Used to modify the edit buttons on the edit form
1.14.0	EditPageBeforeEditChecks	EditPage.php	Allows modifying the edit checks below the textarea in the edit form
1.16.0	EditPageBeforeEditToolbar	EditPage.php	Allows modifying the edit toolbar above the textarea
1.16.0	EditPageCopyrightWarning	EditPage.php	Allow for site and per-namespace customization of contribution/copyright notice.
1.16.0	EditPageGetDiffText	EditPage.php	Allow modifying the wikitext that will be used in "Show changes".
1.16.0	EditPageGetPreviewText	EditPage.php	Allow modifying the wikitext that will be previewed.
1.16.0	EditPageNoSuchSection	EditPage.php	When a section edit request is given for an non-existent section.
1.16.0	EditPageTosSummary	EditPage.php	Give a chance for site and per-namespace customizations of terms of service summary link.
1.11.0	EditSectionLink	EditPage.php	Override the return value of Linker::editSectionLink()
1.11.0	EditSectionLinkForOther	EditPage.php	Override the return value of Linker::editSectionLink()
1.7.0	EmailConfirmed	User.php	When checking that the user's email address is "confirmed"

1.4.0	EmailUser	SpecialEmailuser.php	Occurs whenever the software receives a request to send an email from one user to another
1.17.0	EmailUserCC	SpecialEmailuser.php	Occurs before sending the copy of the email to the author
1.4.0	EmailUserComplete	SpecialEmailuser.php	Occurs after an email has been sent from one user to another
1.17.0	EmailUserForm	SpecialEmailuser.php	Occurs after building the email user form object
1.16.0	EmailUserPermissionsErrors	SpecialEmailuser.php	Retrieves permissions errors for emailing a user
1.19.0	ExemptFromAccountCreationThrottle	SpecialUserlogin.php	To add an exemption from the account creation throttle
1.18.0	ExtensionTypes	SpecialVersion.php	Called when generating the extensions credits, use this to change the tables headers
1.7.0	FetchChangesList	ChangesList.php	Allows extension to modify a recent changes list for a user.
1.13.0	FileDeleteComplete	FileDeleteForm.php	When a file is deleted
1.11.0	FileUpload	filerepo/LocalFile.php	When a file upload occurs
1.13.0	FileUndeleteComplete	SpecialUndelete.php	When a file is undeleted
1.17.0	FormatUserMessage	User.php	Hook to format a message if you want to override the internal formatter.
1.13.0	GetAutoPromoteGroups	Autopromote.php	When determining which autopromote groups a user is entitled to be in.
1.6.0	GetBlockedStatus	User.php	Fired after the user's getBlockStatus is set.
1.13.0	GetCacheVaryCookies	OutputPage.php	get cookies that should vary cache options
1.18.0	GetCanonicalURL	Title.php	Allows to modify fully-qualified URLs used for IRC and e-mail notifications.
1.19.0	GetDefaultSortkey	Title.php	Allows to override what the default sortkey is.
1.6.0	GetFullURL	Title.php	Used to modify fully-qualified URLs used in redirects/export/offsite data
1.6.0	GetInternalURL	Title.php	Used to modify fully-qualified URLs (useful for squid cache purging)
1.17.0	GetIP	ProxyTools.php	Modify the ip of the current user (called only once).
1.12.0	GetLinkColours	Parser.php	modify the CSS class of an array of page links
1.6.0	GetLocalURL	Title.php	Used to modify local URLs as output into page links
1.19.0	GetLocalURL::Article	Title.php	Allows to modify local URLs specifically pointing to article paths without any fancy queries or variants.
1.19.0	GetLocalURL::Internal	Title.php	Allows to modify local URLs to internal pages.
1.18.0	GetMetadataVersion	Generic.php	Allows to modify the image metadata version currently in use.
1.15.0	GetPreferences	Preferences.php	Modify user preferences
1.12.0	getUserPermissionsErrors	Title.php	Add a permissions error when permissions errors are checked for.
1.12.0	getUserPermissionsErrorsExpensive	Title.php	Absolutely the same, but is called only if expensive checks are enabled.
1.14.0	HTMLCacheUpdate::doUpdate	HTMLCacheUpdate.php	After cache invalidation updates are inserted into the job queue

1.13.0	ImageBeforeProduceHTML	Linker.php	Called before producing the HTML created by a wiki image insertion.
1.11.0	ImageOpenShowImageInlineBefore	ImagePage.php	Call potential extension just before showing the image on an image page.
1.16.0	ImagePageAfterImageLinks	ImagePage.php	Called after the image links section on an image page is built.
1.13.0	ImagePageFileHistoryLine	ImagePage.php	Called when a file history line is constructed.
1.13.0	ImagePageFindFile	ImagePage.php	Called when fetching the file associated with an image page.
1.16.0	ImagePageShowTOC	ImagePage.php	Called when the file toc on an image page is generated.
1.17.0	ImportHandleLogItemXMLTag	ImportXMLReader.php	When parsing a XML tag in a log item
1.17.0	ImportHandlePageXMLTag	ImportXMLReader.php	When parsing a XML tag in a page
1.17.0	ImportHandleRevisionXMLTag	ImportXMLReader.php	When parsing a XML tag in a page revision
1.17.0	ImportHandleToplevelXMLTag	ImportXMLReader.php	When parsing a top level XML tag
1.17.0	ImportHandleUploadXMLTag	ImportXMLReader.php	When parsing a XML tag in a file upload
1.11.0	InitPreferencesForm	SpecialPreferences.php	Called at the end of PreferencesForm's constructor
1.16.0	ImgAuthBeforeStream	img_auth.php	Executed before file is streamed to user, but only when using img_auth
1.13.0	InitializeArticleMaybeRedirect	Wiki.php	Called when checking if title is a redirect
1.10.0	InternalParseBeforeLinks	Parser.php	Used to process the expanded wiki code after <nowiki>, HTML-comments, and templates have been treated. Suitable for syntax extensions that want to support templates and comments.
1.16.0	InvalidateEmailComplete	User.php	Called after a user's email has been invalidated successfully.
1.18.0	IRCLineURL	RecentChange.php	When constructing the URL to use in an IRC notification.
1.10.0	IsFileCacheable	Article.php	Allow an extension to disable file caching on pages.
1.9.0	IsTrustedProxy	ProxyTools.php	Allows an extension to set an IP as trusted or not.
1.12.0	isValidEmailAddr	User.php	Override the result of User::isValidEmailAddr()
1.11.0	isValidPassword	User.php	Override the result of User::isValidPassword()
1.6.0	LanguageGetMagic	languages/Language.php	Use this to define synonyms of magic words depending of the language
1.19.0	LanguageGetNamespaces	languages/Language.php	Provide custom ordering for namespaces or remove namespaces.
1.9.0	LanguageGetSpecialPageAliases	languages/Language.php	Use to define aliases of special pages names depending of the language
1.18.0	LanguageGetTranslatedLanguageNames	languages/Language.php	Provide translated language names.
1.14.0	LinkBegin	Linker.php	Used when generating internal and interwiki links in Linker::link()
1.14.0	LinkEnd	Linker.php	Used when generating internal and interwiki links in Linker::link(), just before the function returns a value.
1.13.0	LinkerMakeExternalLink	Linker.php	At the end of Linker::makeExternalLink() just before the return
1.13.0	LinkerMakeExternalImage	Linker.php	At the end of Linker::makeExternalImage() just before the return

1.12.0	LinksUpdate	LinksUpdate.php	At the beginning of LinksUpdate::doUpdate() just before the actual update.
1.12.0	LinksUpdateComplete	LinksUpdate.php	At the end of LinksUpdate::doUpdate() when updating has completed.
1.11.0	LinksUpdateConstructed	LinksUpdate.php	At the end of LinksUpdate() is construction.
1.15.0	ListDefinedTags	ChangeTags.php	When trying to find all defined tags.
1.8.0	LoadAllMessages	MessageCache.php	called by MessageCache::loadAllMessages() to load extensions messages
1.10.1	LoadExtensionSchemaUpdates	maintenance/updaters.inc	Fired when MediaWiki is updated to allow extensions to update the database.
1.13.0	LocalFile::getHistory	LocalFile.php	Called before file history query performed
1.19.0	LogEventsListShowLogExtract	LogEventsList.php	Called before the result of <code>LogEventsList::showLogExtract()</code> is added to <code>OutputPage</code> .
1.10.0	LoginAuthenticateAudit	SpecialUserLogin.php	A login attempt for a valid user account either succeeded or failed. No return data is accepted; this hook is for auditing only.
1.12.0	LogLine	SpecialLog.php	Processes a single log entry on <code>Special:Log</code>
1.6.0	LogPageActionText	Setup.php	Strings used by wfMsg as a header. DEPRECATED: Use <code>\$wgLogActions</code>
1.5.0	LogPageLogHeader	Setup.php	Strings used by wfMsg as a header. DEPRECATED: Use <code>\$wgLogHeaders</code>
1.5.0	LogPageLogName	Setup.php	Name of the logging page(s). DEPRECATED: Use <code>\$wgLogNames</code>
1.5.0	LogPageValidTypes	Setup.php	Action being logged. DEPRECATED: Use <code>\$wgLogTypes</code>
1.6.0	MagicWordMagicWords	MagicWord.php	Allows extensions to define new magic words. DEPRECATED: Use <code>LanguageGetMagic</code> instead.
1.6.0	MagicWordwgVariableIDs	MagicWord.php	Allows extensions to add Magic Word IDs.
1.18.0	MaintenanceRefreshLinksInit	maintenance/refreshLinks.php	Before executing the refreshLinks.php maintenance script.
1.14.0	MakeGlobalVariablesScript	Skin.php	called right before <code>Skin::makeVariablesScript()</code> is executed
1.6.0	MarkPatrolled	Article.php	Called before an edit is marked patrolled
1.6.0	MarkPatrolledComplete	Article.php	Called after an edit is marked patrolled
1.7.0	MathAfterTexvc	Math.php	After texvc is executed when rendering mathematics
1.12.0	MediaWikiPerformAction	Wiki.php	Override <code>MediaWiki::performAction()</code> .
1.15.0	MessageCacheReplace	MessageCache.php	When a message page is changed. Useful for updating caches.
1.16.0	MessageNotInMwNs	MessageCache.php	When trying to get a message that isn't found in the MediaWiki namespace
1.5.7	MessagesPreLoad	MessageCache.php	Occurs when loading a message from the database
1.16.0	ModifyExportQuery	Export.php	Modify the query used by the exporter.
1.6.0	MonoBookTemplateToolboxEnd	skins/Monobook.php	Called by Monobook skin after toolbox links have been rendered (useful for adding more)

1.15.0	NewDifferenceEngine	diff/DifferenceEngine.php	Called when a new DifferenceEngine object is made.
1.13.0	NewRevisionFromEditComplete	(Various files)	Called when a revision was inserted due to an edit
1.13.0	NormalizeMessageKey	MessageFunctions.php	Called before the software gets the text of a message
1.14.0	OldChangesListRecentChangesLine	ChangesList.php	Customize entire Recent Changes line
1.13.0	OpenSearchUrls	opensearch_desc.php	Called when constructing the OpenSearch description XML.
1.16.0	OtherBlockLogLink	SpecialBlockip.php	Get links to the block log from extensions which blocks users and/or IP addresses too
1.6.0	OutputPageBeforeHTML	OutputPage.php	Called after the page has been rendered, but before the HTML is displayed.
1.17.0	OutputPageBodyAttributes	OutputPage.php	Called when OutputPage::headElement() is creating the body tag.
1.14.0	OutputPageCheckLastModified	OutputPage.php	When checking if the page has been modified since the last visit
1.13.0	OutputPageMakeCategoryLinks	OutputPage.php	Called when links are about to be generated for the page's categories.
1.8.0	OutputPageParserOutput	OutputPage.php	after adding a parserOutput to \$wgOut
1.19.0	PageContentLanguage	Title.php	Allows changing the page content language and the direction depending on that language.
1.10.0	PageHistoryBeforeList	PageHistory.php	When a history page list is about to be constructed.
1.10.0	PageHistoryLineEnding	PageHistory.php	Right before the end >li> is added to a history line
1.13.0	PageHistoryPager::getQueryInfo	PageHistory.php	When a history pager query parameter set is constructed
1.6.0	PageRenderingHash	User.php	Alter the parser cache option hash key
1.5.0	ParserAfterStrip	Parser.php	Same as ParserBeforeStrip
1.5.0	ParserAfterTidy	Parser.php	Used to add some final processing to the fully-rendered page output
1.6.0	ParserBeforeInternalParse	Parser.php	called at the beginning of Parser::internalParse()
1.5.0	ParserBeforeStrip	Parser.php	Used to process the raw wiki code before any internal processing is applied
1.5.0	ParserBeforeTidy	Parser.php	Used to process the nearly-rendered html code for the page (but before any html tidying occurs)
1.6.0	ParserClearState	Parser.php	called at the end of Parser::clearState()
1.12.0	ParserFirstCallInit	Parser.php	called when the parser initialises for the first time
1.6.0	ParserGetVariableValueSwitch	Parser.php	called when the parser needs the value of a custom magic word
1.6.0	ParserGetVariableValueTs	Parser.php	use this to change the value of the time for the {{LOCAL...}} magic word
1.6.0	ParserGetVariableValueVarCache	Parser.php	use this to change the value of the variable cache or return false to not use it
1.12.0	ParserLimitReport	Parser.php	called at the end of Parser::parse() when the parser will include comments about size of the text parsed
1.12.0	ParserMakeImageParams	Parser.php	called at the end of Parser::makeImage(). Alter the parameters used to generate an image.
1.19.0	ParserSectionCreate	Parser.php	Called each time the parser creates a document section from wikitext.

1.6.0	ParserTestParser	maintenance/Parser.inc	called when creating a new instance of Parser in maintenance/parserTests.inc
1.10.0	ParserTestTables	maintenance/parserTests.inc	Alter the list of tables to duplicate when parser tests are run. Use when page save hooks require the presence of custom tables to ensure that tests continue to run properly.
1.18.0	PerformRetroactiveAutoblock	Block.php	Called before a retroactive autoblock is applied to a user.
1.6.0	PersonalUrls	SkinTemplate.php	Alter the user-specific navigation links (e.g. "my page, my talk page, my contributions" etc).
1.9.0	PingLimiter	User.php	Allows extensions to override the results of User::pingLimiter()
1.9.0	PreferencesUserInformationPanel	SpecialPreferences.php	Add HTML bits to user information list in preferences form
1.12.0	PrefixSearchBackend	PrefixSearch.php	Override the title prefix search used for OpenSearch and AJAX search suggestions.
1.11.0	PrefsEmailAudit	SpecialPreferences.php	called when user changes his email address
1.11.0	PrefsPasswordAudit	SpecialPreferences.php	called when user changes his password
1.16.0	ProtectionForm::buildForm	ProtectionForm.php	Called after all protection type fieldsets are made in the form
1.16.0	ProtectionForm::save	ProtectionForm.php	Called when a protection form is submitted
1.16.0	ProtectionForm::showLogExtract	ProtectionForm.php	Called after the protection log extract is shown
1.10.0	RawPageViewBeforeOutput	RawPage.php	Right before the text is blown out in action=raw
1.8.0	RecentChange_save	RecentChange.php	Called after a "Recent Change" is committed to the DB
1.11.0	RenderPreferencesForm	SpecialPreferences.php	Called at the end of PreferencesForm::mainPrefsForm
1.11.0	ResetPreferences	SpecialPreferences.php	Called at the end of PreferencesForm::resetPrefs
1.17.0	ResourceLoaderGetConfigVars	ResourceLoaderStartUpModule.php	Called right before ResourceLoaderStartUpModule::getConfig() returns.
1.18.0	ResourceLoaderGetStartupModules	ResourceLoaderStartUpModule.php	Run once the startup module is being generated.
1.17.0	ResourceLoaderRegisterModules	ResourceLoader.php	Allows registering of modules with ResourceLoader.
1.11.0	RevisionInsertComplete	Revision.php	Called after a revision is inserted into the DB
1.11.0	SavePreferences	SpecialPreferences.php	Called at the end of PreferencesForm::savePreferences; returning false prevents the preferences from being saved.
1.16.0	SearchableNamespaces	SearchEngine.php	An option to modify which namespaces are searchable.
1.16.0	SearchEngineReplacePrefixesComplete	SearchEngine.php	Run after SearchEngine::replacePrefixes().
1.12.0	SearchGetNearMatch	SearchEngine.php	An extra chance for exact-title-matches in "go" searches.
1.16.0	SearchGetNearMatchBefore	SearchEngine.php	Perform exact-title-matches in "go" searches before the normal operations.
1.16.0	SearchGetNearMatchComplete	SearchEngine.php	A chance to modify exact-title-matches in "go" searches.
1.10.0	SearchUpdate	SearchUpdate.php	Prior to search update completion
1.14.0	SetupAfterCache	Setup.php	Called in Setup.php, after cache objects are set
1.17.0	SetupUserMessageArticle	User.php	Called in User::leaveUserMessage() before anything has been posted to the article.
1.16.0	ShowMissingArticle	Article.php	Called when generating the output for a non-existent page

1.11.0	ShowRawCssJs	Article.php	Customise the output of raw CSS and JavaScript in page views
1.16.0	ShowSearchHitTitle	SpecialSearch.php	Customise display of search hit title/link.
1.7.0	SiteNoticeAfter	GlobalFunctions.php	After site notice is determined
1.7.0	SiteNoticeBefore	GlobalFunctions.php	Before site notice is determined
1.11.0	SkinAfterBottomScripts	Skin.php	At the end of Skin::bottomScripts()
1.14.0	SkinAfterContent	Skin.php	Allows extensions to add text after the page content and article metadata.
1.14.0	SkinBuildSidebar	Skin.php	At the end of Skin::buildSidebar()
1.16.0	SkinCopyrightFooter	Skin.php	Allow for site and per-namespace customization of copyright notice.
1.17.0	SkinGetPoweredBy	Skin.php	Called when generating the code used to display the "Powered by MediaWiki" icon.
1.12.0	SkinSubPageSubtitle	Skin.php	Called before the list of subpage links on top of a subpage is generated
1.6.0	SkinTemplateBuildContentActionUrlsAfterSpecialPage	SkinTemplate.php	after the single tab when showing a special page
1.6.0	SkinTemplateBuildNavUrlsNav_urlsAfterPermalink	SkinTemplate.php	after creating the "permanent link" tab
1.5.0	SkinTemplateContentActions	SkinTemplate.php	Alter the "content action" links in SkinTemplates
1.16.0	SkinTemplateNavigation	SkinTemplate.php	Alter the structured navigation links in SkinTemplate
1.10.0	SkinTemplateOutputPageBeforeExec	SkinTemplate.php	Before SkinTemplate::outputPage() starts page output
1.6.0	SkinTemplatePreventOtherActiveTabs	SkinTemplate.php	use this to prevent showing active tabs
1.6.0	SkinTemplateSetupPageCss	SkinTemplate.php	use this to provide per-page CSS
1.18.0	SkinTemplateNavigation::SpecialPage	SkinTemplate.php	Called on special pages after the special tab is added but before variants have been added
1.12.0	SkinTemplateTabAction	SkinTemplate.php	Override SkinTemplate::tabAction().
1.6.0	SkinTemplateTabs	SkinTemplate.php	called when finished to build the actions tabs
1.13.0	SkinTemplateToolboxEnd	skins/Monobook.php, skins/Modern.php	Called by SkinTemplate skins after toolbox links have been rendered (useful for adding more)
1.18.0	SkinTemplateNavigation::Universal	SkinTemplate.php	Called on both content and special pages after variants have been added
1.15.0	SoftwareInfo	SpecialVersion.php	Called by Special:Version for returning information about the software
1.5.0	SpecialContributionsBeforeMainOutput	SpecialContributions.php	Before the form on Special:Contributions
1.13.0	SpecialListusersDefaultQuery	SpecialListusers.php	Called right before the end of UsersPager::getDefaultQuery()
1.13.0	SpecialListusersFormatRow	SpecialListusers.php	Called right before the end of UsersPager::formatRow()
1.13.0	SpecialListusersHeader	SpecialListusers.php	Called before closing the <fieldset> in UsersPager::getPageHeader()
1.13.0	SpecialListusersHeaderForm	SpecialListusers.php	Called before adding the submit button in UsersPager::getPageHeader()
1.13.0	SpecialListusersQueryInfo	SpecialListusers.php	Called right before the end of UsersPager::getQueryInfo()
1.6.0	SpecialMovepageAfterMove	SpecialMovepage.php	Called after a page is moved.
1.18.0	SpecialNewPagesFilters	SpecialNewpages.php	Called after building form options at NewPages.

1.7.0	SpecialPage_initList	SpecialPage.php	Called when the list of Special Pages is populated. Hook gets a chance to change the list to hide/show pages
1.6.0	SpecialPageExecuteAfterPage	SpecialPage.php	called after executing a special page
1.6.0	SpecialPageExecuteBeforeHeader	SpecialPage.php	called before setting the header text of the special page
1.6.0	SpecialPageExecuteBeforePage	SpecialPage.php	called after setting the special page header text but before the main execution
1.18.0	SpecialPasswordResetOnSubmit	SpecialPasswordReset.php	Called when executing a form submission on Special:PasswordReset.
1.16.0	SpecialRandomGetRandomTitle	SpecialRandompage.php	Modify the selection criteria for Special:Random
1.18.0	SpecialRecentChangesFilters	SpecialRecentchanges.php	Called after building form options at RecentChanges
1.13.0	SpecialRecentChangesPanel	SpecialRecentchanges.php	called when building form options in SpecialRecentChanges
1.13.0	SpecialRecentChangesQuery	SpecialRecentchanges.php	called when building sql query for SpecialRecentChanges
1.19.0	SpecialSearchCreateLink	SpecialSearch.php	Called when making the message to create a page or go to the existing page.
1.6.0	SpecialSearchNogomatch	SpecialSearch.php	called when user clicked the "Go" button but the target doesn't exist
1.13.0	SpecialSearchNoResults	SpecialSearch.php	called before search result display when there are no matches
1.19.0	SpecialSearchPowerBox	SpecialSearch.php	The equivalent of SpecialSearchProfileForm for the advanced form, a.k.a. power search box
1.18.0	SpecialSearchProfileForm	SpecialSearch.php	allows modification of search profile forms
1.16.0	SpecialSearchProfiles	SpecialSearch.php	allows modification of search profiles
1.13.0	SpecialSearchResults	SpecialSearch.php	called before search result display when there are matches
1.18.0	SpecialSearchSetupEngine	SpecialSearch.php	allows passing custom data to search engine
1.16.0	SpecialStatsAddExtra	SpecialStatistics.php	Can be used to add extra statistic at the end of Special:Statistics
1.16.0	SpecialUploadComplete	SpecialUpload.php	Called after successfully uploading a file from Special:Upload
1.6.0	SpecialVersionExtensionTypes	SpecialVersion.php	called when generating the extensions credits, use this to change the tables headers
1.18.0	SpecialWatchlistFilters	SpecialWatchlist.php	Called after building form options at Watchlist.
1.14.0	SpecialWatchlistQuery	SpecialWatchlist.php	Called when building sql query for SpecialWatchlist
1.18.0	TestCanonicalRedirect	Wiki.php	Called when about to force a redirect to a canonical URL for a title when we have no other parameters on the URL.
1.14.0	TitleArrayFromResult	TitleArray.php	called when creating an TitleArray object from a database result
1.16.0	TitleGetRestrictionTypes	Title.php	Allows to modify the types of protection that can be applied.
1.19.0	TitlesCssOrJsPage	Title.php	Called when determining if a page is a CSS or JS page.
1.19.0	TitlesMovable	Title.php	Called when determining if it is possible to move a page.
1.19.0	TitlesWikitextPage	Title.php	Called when determining if a page is a wikitext or should be handled by seperate handler (via ArticleViewCustom).
1.4.0	TitleMoveComplete	Title.php	Occurs whenever a request to move an article is completed

1.18.0	UndeleteForm::showHistory	SpecialUndelete.php	Called in UndeleteForm::showHistory, after a PageArchive object has been created but before any further processing is done.
1.18.0	UndeleteForm::showRevision	SpecialUndelete.php	Called in UndeleteForm::showRevision, after a PageArchive object has been created but before any further processing is done.
1.18.0	UndeleteForm::undelete	SpecialUndelete.php	Called in UndeleteForm::undelete, after checking that the site is not in read-only mode, that the Title object is not null and after a PageArchive object has been constructed but before performing any further processing.
1.9.0	UndeleteShowRevision	SpecialUndelete.php	called when showing a revision in Special:Undelete
1.4.0	UnknownAction	Wiki.php	Used to add new query-string actions
1.4.0	UnwatchArticle	Article.php	Occurs whenever the software receives a request to unwatch an article
1.4.0	UnwatchArticleComplete	Article.php	Occurs after the unwatch article request has been processed
1.16.0	UploadCreateFromRequest	UploadBase.php	When UploadBase::createFromRequest has been called.
1.6.4	UploadComplete	SpecialUpload.php	Called when a file upload has completed.
1.16.0	UploadFormInitDescriptor	SpecialUpload.php	Occurs after the descriptor for the upload form as been assembled.
1.16.0	UploadFormSourceDescriptors	SpecialUpload.php	Occurs after the standard source inputs have been added to the descriptor.
1.9.0	UploadForm:BeforeProcessing	SpecialUpload.php	Called just before the file data (for example description) are processed, so extensions have a chance to manipulate them.
1.9.0	UploadForm:initial	SpecialUpload.php	Called just before the upload form is generated
1.6.0	UploadVerification	SpecialUpload.php	Called when a file is uploaded, to allow extra file verification to take place (deprecated)
1.17.0	UploadVerifyFile	SpecialUpload.php	Called when a file is uploaded, to allow extra file verification to take place (preferred)
1.18.0	UserAddGroup	User.php	Called when adding a group to a user.
1.13.0	UserArrayFromResult	UserArray.php	called when creating an UserArray object from a database result
1.6.0	userCan	Title.php	To interrupt/advise the "user can do X to Y article" check
1.12.0	UserCanSendEmail	User.php	To override User::canSendEmail() permission check
1.5.7	UserClearNewTalkNotification	User.php	called when clearing the "You have new messages!" message
1.14.0	UserComparePasswords	User.php	Called when checking passwords
1.6.0	UserCreateForm	SpecialUserlogin.php	Change to manipulate the login form
1.14.0	UserCryptPassword	User.php	Called when hashing a password
1.11.0	UserEffectiveGroups	User.php	Called in User::getEffectiveGroups()
1.13.0	UserGetAllRights	User.php	After calculating a list of all available rights
1.18.0	UserGetDefaultOptions	User.php	Called after fetching the core default user options.
1.13.0	UserGetEmail	User.php	Called when getting an user email address

1.13.0	UserGetEmailAuthenticationTimestamp	User.php	Called when getting the timestamp of email authentication
1.11.0	UserGetImplicitGroups	User.php	Called in User::getImplicitGroups()
1.18.0	UserGetLanguageObject	User.php	Called when getting user's interface language object.
1.14.0	UserGetReservedNames	User.php	Allows to modify \$wgReservedUsernames at run time.
1.11.0	UserGetRights	User.php	Called in User::getRights()
1.16.0	UserIsBlockedFrom	User.php	Check if a user is blocked from a specific page (for specific block exemptions).
1.14.0	UserIsBlockedGlobally	User.php	Check if user is blocked on all wikis.
1.14.0	UserLoadAfterLoadFromSession	User.php	Called to authenticate users on external/environmental means; occurs after session is loaded
1.13.0	UserLoadDefaults	User.php	Called when loading a default user
1.15.0	UserLoadFromDatabase	User.php	Called when loading a user from the database
1.13.0	UserLoadFromSession	User.php	Called to authenticate users on external/environmental means
1.15.0	UserLoadOptions	User.php	Called when user options/preferences are being loaded from the database
1.4.0	UserLoginComplete	SpecialUserlogin.php	Occurs after a user has successfully logged in
1.6.0	UserLoginForm	SpecialUserlogin.php	Change to manipulate the login form
1.4.0	UserLogout	SpecialUserlogin.php	Occurs when the software receives a request to log out
1.4.0	UserLogoutComplete	SpecialUserlogin.php	Occurs after a user has successfully logged out
1.14.0	User::mailPasswordInternal	SpecialUserlogin.php	Before creation and mailing of a user's new temporary password
1.18.0	UserRemoveGroup	User.php	Called when removing a group from an user.
1.5.7	UserRetrieveNewTalks	User.php	called when retrieving "You have new messages!" message(s)
1.6.0	UserRights	SpecialUserrights.php	Called after a user's group memberships are changed
1.14.0	UserRightsChangeableGroups	SpecialUserrights.php	Called after the list of groups that can be manipulated via Special:UserRights is populated, but before it is returned.
1.15.0	UserSaveOptions	User.php	Called just before saving user preferences/options
1.13.0	UserSaveSettings	User.php	Called when saving user settings
1.13.0	UserSetCookies	User.php	Called when setting user cookies
1.13.0	UserSetEmail	User.php	Called when changing user email address
1.13.0	UserSetEmailAuthenticationTimestamp	User.php	Called when setting the timestamp of email authentication
1.8.0	UserToggles	User.php	Called before returning "user toggle names"
1.18.0	WantedPages::getQueryInfo	SpecialWantedPages.php	Called in WantedPagesPage::getQueryInfo(), can be used to alter the SQL query which gets the list of wanted pages.
1.15.0	WantedPages::getSQL	SpecialWantedPages.php	Called in WantedPagesPage::getSQL(), can be used to alter the SQL query which gets the list of wanted pages
1.4.0	WatchArticle	Article.php	Occurs whenever the software receives a request to watch an article
1.4.0	WatchArticleComplete	Article.php	Occurs after the watch article request has been processed

1.17.0	WatchlistEditorBuildRemoveLine	WatchlistEditor.php	Occurs when building remove lines in Special:Watchlist/edit
1.19.0	WebRequestGetPathInfoRequestURI	WebRequest.php	Occurs while extracting path info from REQUEST_URI.
1.19.0	wfShellMaintenanceCmd	GlobalFunctions.php	Called when generating a shell-escaped command line string to run a maintenance script.
1.15.0	WikiExporter::dumpStableQuery	Export.php	Get the SELECT query for "stable" revisions dumps
1.6.0	wgQueryPages	QueryPage.php	called when initialising \$wgQueryPages
1.16.0	XmlDumpWriterOpenPage	Export.php	Called at the end of XmlDumpWriter::openPage, to allow extra metadata to be added.
1.16.0	XmlDumpWriterWriteRevision	Export.php	Called at the end of a revision in an XML dump, to add extra metadata.
1.18.0	XMPGetInfo	media/XMPInfo.php	Called when obtaining the list of XMP tags to extract.
1.18.0	XMPGetResults	media/XMP.php	Called just before returning the results array of parsing xmp data.

Hooks grouped by version

To see hooks grouped by version go to the table above and click the arrow symbol in the version table header.

References

- [1] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/Hooks.php?view=markup>
- [2] <http://www.php.net/manual/en/function.call-user-func-array.php>
- [3] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/docs/hooks.txt?view=markup>

Extension hook registry

The Hooks mechanism in MediaWiki is a powerful tool for extension authors to modify the behavior of MediaWiki while minimizing, if not eliminating, the need to hack the codebase. A logical extension of this idea is to place hooks in extensions themselves, in order to allow other developers to extend the extensions. This page provides a listing and documentation for hooks provided by extension authors. To document your hook:

1. create a page named after the hook under your extension, e.g. `Extension:MyExtension/MyHook`. *Note: this naming convention allows defaults and category links in `Template:Extension` and `Template:SeeCustomHook` to work properly*
2. place the `Template:Extension` in your page and fill in the parameter values
3. add a row for your hook in the table below

Extension hooks by parent extension

Parent extension	Version Extension	Revision SVN	Hook	Description
ArticleComments	0.4	r73901	ArticleCommentsSpamCheck	Called on comment submission for spam detection.
Backup			Backup	Hook event called when a page change is effected
CategoryHook	0.2		CategoryHook	Adds a hook allowing articles to be added to additional categories based on wikitext content.
CentralAuth		1.13	CentralAuthWikiList	Get the list of wikis in which CentralAuth is active
ContactPage			ContactForm	Can be used to manipulate most of the values related to the contact message: the sender, the receiver, the subject and the actual contact message text.
ContactPage		r60638	ContactFormBeforeMessage	Can be used to add new fields to Special:Contact. Two parameters, \$contactForm (Object; instance of EmailContactForm class) and &\$form (HTML used to generate the form).
ContactPage			ContactFormComplete	Takes the same parameters as ContactForm hook does.
DatabaseFetchObject	0.1		DatabaseFetchObject	Hook for checking permissions in any context where database page contents are retrieved.
EmailDomainCheck	0.1		EmailDomainCheck	Only allows registrations from a specific email domain
GoogleNewsSitemap			GoogleNewsSitemap::Query	Modifies the query used by GoogleNewsSitemap (used for flagged revs support)
ImagePageEx			ImagePageEx	Hooks for trapping delete related changes in NS_IMAGE
LDAP Authentication	1.1c+		SetUsernameAttributeFromLDAP	Allows hook function to set the LDAP username from various attributes returned by a query.
NewUserMessage	3.1		CreateNewUserMessage	Hook for modules that allows them to override the default new user message.
PagesOnDemand	0.1		PagesOnDemand	Hook for modules that create new pages from templates when links to nonexistent pages are followed
Renameuser		1.11	RenameUserAbort	Allow other extensions to abort a user rename
Renameuser		1.11	RenameUserComplete	After a successful user rename
Renameuser		1.13	RenameUserPreRename	Before renaming a user
Renameuser		r55189	RenameUserSQL	In the constructor of RenameuserSQL class, allows adding tables to the array of tables that Renameuser should update on rename
Renameuser		1.13	RenameUserWarning	Get warnings when renaming a user
Shibboleth Authentication			ShibUpdateUser	Hook event called before user information gets updated on login (for setting the name/mail from Shibboleth)
UserMerge	1.6.21	1.19 r89147	DeleteAccount	Trigger update actions elsewhere (e.g. in OpenID) when UserMerge has deleted an account
UserMerge	1.6.21	1.19 r89147	MergeAccountFromTo	Trigger update actions elsewhere (e.g. in OpenID) when UserMerge has merged one account to another
SphinxSearch	0.7.2	r73342	SphinxSearchBeforeResults	Control various search parameters before search client is instantiated
SphinxSearch	0.7.2	r73342	SphinxSearchBeforeQuery	Control search term and sphinx client before search is executed

Extensions:	Category • All • Requests • Tag extensions • Extension Matrix • Extensions FAQ • Extension hook registry • Extension namespace registry
-------------	---

Resource Loader

ResourceLoader

WMF engineering project

ResourceLoader

A feature to improve the load times for JavaScript and CSS in MediaWiki.

Group:	Features
Start:	2010-06
End:	
Lead:	Alolita Sharma
Team:	Roan Kattouw, Timo Tijhof

ResourceLoader is a JavaScript/CSS delivery optimizing system for MediaWiki that is available in MediaWiki version 1.17 and later. Its goal is to improve MediaWiki's front-end performance.

Extension Developers: Please integrate ResourceLoader support in your extensions. While extensions that do not support ResourceLoader should still function in MediaWiki 1.17, there may be some issues.

About ResourceLoader

On Wikimedia wikis, every page view includes hundreds of kilobytes of JavaScript. In many cases, some or all of this code goes unused due to browser support or because users do not make use of the features on the page. In these cases, bandwidth, and loading time spent on downloading, parsing and executing JavaScript code is wasted. This is especially true when users visit MediaWiki sites using older browsers like Internet Explorer 6, where almost all features are unsupported, and parsing and executing JavaScript is extremely slow.



ResourceLoader solves this problem by loading resources on demand and only for browsers that can run them. It also improves client-side performance by minifying JavaScript and CSS code (which reduces the code's size and parsing time), batch loading resources (which reduces the number of requests made), and optionally embedding images as data URIs (which can further reduce the number of requests made).

Documentation

- **Migration guide (users)**
- **Migration guide for extension developers** – a guide to making extensions to work with ResourceLoader.
- **Developing with ResourceLoader** – notes on configuring your development environment and on switching ResourceLoader between development and production modes.
- **Default modules** – an overview of all default modules and their methods, supported with documentation and examples.
- **Vocabulary** – the vocabulary used in ResourceLoader (such as Loader, Module, Requirement, etc.)
- **Requirements** – requirements gathered from a variety of sources during the planning stage.
- **JavaScript Deprecations** – JavaScript features that are planned to be replaced with modern equivalents.

Project information

- **Version 1 Design Specification** – the design specifications developed and maintained throughout the development process.
- **Version 2 Design Specification** – the design specifications begin developed for the next version.

Todo and tasks

- Tasks - Random uncategorized todo list
- Task management (V1) – **tracking bug for V1**
- Task management (V2) – **tracking bug for V2**
- /status

ResourceLoader/Requirements

Requirement	Features	Benefits	Issues
Debugging			
Support direct resource inclusion	Allows JS resources to be included individually and without being run through the loader software	JS resources can be more easily debugged because file names and line numbers correspond to the actual JS resources	Requires L10n fallbacks to be embedded in JS resources
Support stand-alone use of loader	Allows the script loader system to be integrated into other frameworks or used as a stand-alone service	Development efforts can be shared with non-MediaWiki developers, and those developers' efforts can benefit us	Makes MediaWiki integration difficult, makes code more foreign to MediaWiki developers
Useful information on loading failure	Provides useful, verbose information about failures to load resources	Makes development easier	Requires more code to be written
Packaging			
Remove debug statements	Allows JS resources to contain debug information which is stripped out before being sent to the client	Debug information can remain in JS resources for future use	More moving parts to break
JS/CSS Minification	Strips whitespace and comments from JS and CSS resources	Reduces the amount of data sent to the client	Makes debugging more difficult
JS Closure Compilation	Simplifies and aggressively optimizes JS functions	Reduces the amount of data sent to the client, and improves the performance of many JS functions	Makes debugging very difficult

CSS Janus transformation	Flips CSS rules to make LTR stylesheets appropriate for RTL	No need to keep separate files around for this if we can apply this transformation in PHP along with the others	Makes it slightly more difficult to figure out what the RTL version of a given CSS file looks like
Resource batching / concatenation	Multiple resources can be included in a single request	Reduces the number of requests from clients	Potentially increases cache size, potentially causes double inclusion of resources
Parameterize batched loading	Set a URL param to indicate we need resources for a certain combination of prefs/logged-in-ness rather than detecting this on the server	Eliminates the need for uncacheable-for-logged-in-users Vary: Cookie responses	Increases the number of URL variants
Web-server-level resource caching	Caches processed (localized, minified, etc.) scripts on the server	Makes it faster to batch resources in different combination	Caching is not free, it takes time and space
Use short client-side expiry times	Tells clients to refresh cached resources in short intervals (~1h)	Eliminates the need to expire HTML (expensive) to force clients to refresh resources	Requires that RL requests be cheap, so they should support 304 and be cached in Squid
Use highest-timestamp versioning scheme	Version each RL request with the highest last-modified timestamp of the resources involved, formatted in ISO 8601 and rounded to 10s	No more manual style versions. 10s rounding and human-readable format allows faster and easier purging of bad resources from Squid	May cause conflicts when new JS is served with old HTML, need to code around that in JS
In-place localization	Includes localized messages directly within JS resources	Allows i18n of JS software without adding additional requests	Mixes i18n work with development work
Localization API	Makes messages available on demand individually or in groups	Provides yet another option for getting L10n to the client	Very inefficient way to get L10n to the client
Separate message resource system	Generates JS blobs for l10n separately, stores them in the DB	Only relevant JS is regenerated when a message changes	Makes stand-alone usage more difficult
Export user preferences to JS			
CSS grouping (without JS)			
CSS includes registered in JS namespace	Allows JS resources to check existence of CSS resources and require CSS resources to be loaded before proceeding	Fewer CSS resources need to be included initially and modules can safely avoid re-loading already-loaded CSS resources	Requires more code to be written
Resource buckets	Allows resources to be grouped into buckets such as "all", "view", "edit", etc.	Reduces number of individual requests needed	Potentially too many things will get put in these buckets
Runtime loading of JS and CSS	Allows resources to be included dynamically	Reduces the initial payload sent to clients	Potentially presents delay between user action and resource availability
Module based loading	Resources are identified by module names	Provides a standard way of expressing requirements and checking what resources are already loaded	
Module loaders	Modules each have associated loader functions which are executed to load the module	Provides more flexibility in loading	Increases the amount of code sent to the client and level of complexity in development
Configurable URL scheme	Allows flexibility in request URL formats	Makes using a localhost, web-server or CDN a matter of configuration	

Allow version aliasing	In the case that a buggy version of a script is released, a version redirect can be put in place	Avoids purging PHP generated HTML to resolve JS bugs	Potentially introduces level of indirection causing negative performance impact. Not needed if scripts are versioned and cached in Squid
Dependency order preservation	Allows dependencies to automatically be included in a specific order if needed	Makes dependencies simpler to use	May make dependency specification more complex
Structure			
Clear division of libraries	Makes JS resources communicate with each other using well documented APIs	Improves the re-usability of JS resources	Takes more time to develop and document
Control a specific JS namespace	Specifies a unified naming convention and access point for JS resources	Improves maintainability and reduces difficulty to learn the system	Requires porting existing code to conform
Control naming between PHP and JS resources	Specifies clear naming between PHP extensions and JS modules	Improves maintainability and reduces difficulty to learn the system	Requires porting existing code to conform
Package-level dependency resolution	JS modules can form packages, which can then be required/included as a whole	Reduces complexity of specifying requirements making development easier	Reduces specificity of requirements potentially causing resources to be loaded that will not be used
Map files, buckets and dependencies in PHP	Have an associative array structure in PHP that defines objects, which optionally contain a file and optionally have dependencies	Makes for fewer distinct URLs (?), which eases caching and purging	Limited support for JS-based conditions, would have to be done through parameterization, cannot scale to user scripts
Division between loading and execution	JS modules can be loaded on demand, and executed independently	Makes dependencies simpler to use	Makes writing modules more complex
Allow explicit re-loading	Resources are not loaded more than once by default, but can be re-loaded if desired	Makes overwriting/re-loading possible/easier	
jQuery should be the primary way of extending functionality			
jQuery plug-ins should be centrally organized	All jQuery plug-ins should be committed to a central place, with version numbers in their names	Makes sharing plug-ins easier / more likely to happen and everyone benefits from bug-fixes equally	Causes issues with maintaining extensions or if people cloud the plug-in folder with low-quality plug-ins
JS module version conflict management	Makes module version information available to JS resources	Makes it possible to migrate to newer versions of plug-ins over time	Sends more data to the client, adds complexity to configuration
Utilities			
JS localization functions	JS facilities for handling 18n functionality such as {{PLURAL}} and \$ replacement	Messages can be more flexible and reusable	Must be maintained in parallel with PHP 18n functions, limitations must be well understood by translators

Sources

This document is a synthesis of the following sources

- Michael Dale
- Tim Starling
- Neil Kandalgaonkar

Documentation	Migration guide (users) · Migration guide (developers) · Developing with ResourceLoader · Default modules · Vocabulary
Project information	Status updates · Version 1 Design Specification (tasks) · Version 2 Design Specification (tasks) · Requirements
Other	JavaScript Deprecations

ResourceLoader/Migration guide for extension developers

Most code written prior to MediaWiki 1.17, the introduction of ResourceLoader, should continue to work, there are some issues that are specific to the architecture of the system which may need to be resolved.

Crash-course: How to use ResourceLoader

Review `resources/Resources.php` ^[1] for more examples of module registration and `includes/OutputPage.php` ^[2] for more examples of how to add modules to a page.

If you are porting your extension to ResourceLoader, please add it to the list of ResourceLoader compatible extensions. The extensions on this list should also be useful as examples.

For now let's start with how to register a module:

Registering a module

This section is a work in progress, updating since r77011 ^[3]

To make your resources available to the loader, you will need to register them as a module, telling ResourceLoader what you want to make available and where the files are.

```
/* MyExtension.php */

$wgResourceModules['ext.myExtension'] = array(
    // JavaScript and CSS styles. To combine multiple file, just list
    them as an array.
    'scripts' => array( 'js/ext.myExtension.core.js',
'js/ext.myExtension.foobar.js' ),
    'styles' => 'css/ext.myExtension.css',

    // When your module is loaded, these messages will be available
    through mw.msg()
    'messages' => array( 'myextension-hello-world',
'myextension-goodbye-world' ),
```

```
// If your scripts need code from other modules, list their
identifiers as dependencies
// and ResourceLoader will make sure they're loaded before you.
// You don't need to manually list 'mediawiki' or 'jquery', which
are always loaded.
'dependencies' => array( 'jquery.ui.datepicker' ),

// ResourceLoader needs to know where your files are; specify
your
// subdir relative to "/extensions" (or $wgExtensionAssetsPath)
'localBasePath' => dirname( __FILE__ ),
'remoteExtPath' => 'MyExtension'
);
```

Notice that we only list the dependency we directly need. Other modules such as **jquery** and **jquery.ui** will be automatically loaded to support **jquery.ui.datepicker**. Also notice that because this is a module provided by and related to an extension, the module name begins with "ext."

Tip: Pass single resources as a string. Pass multiple resources as an array of strings.

Adding a module to the page

While building the page, if you want a module to be loaded before document ready, you need tell the `OutputPage` object to add one or more modules to the page.

```
$wgOut->addModules( 'ext.myExtension' );
```

Tip: Provide `addModules` with an array of module names if you want to load more than one.

Loading a module dynamically

If the page is loaded, it's still not too late to get your module loaded on the client. You will just need to ask the **mw.loader** object to get it for you.

```
mw.loader.using( 'ext.myExtension', function() {
    /* Do something with ext.myExtension, now that it's ready to use
*/
} );
```

Tip: If you just want to load the module, and don't need to run any code when it arrives, you can just use **mw.loader.load**('ext.myExtension').

Internationalization

You can access messages specified in your resource module using the `mw.msg()` method. Example:

```
alert ( mw.msg( 'myextension-hello-world' ) );
```

`mw.msg()` can accept multiple arguments, with additional arguments being passed to the message function named in the first argument (exactly in the same way as with server-side message functions):

```
alert ( mw.msg( 'myextension-about-me', myName, myTitle ) );
```

Inline JavaScript

In previous versions of MediaWiki, nearly all JavaScript resources were added in the head of the document. With the introduction of ResourceLoader, JavaScript resources are now loaded at the bottom of the body.

The motivation for this change has to do with the fact that most browsers will wait for scripts to load and execute before continuing to load the page. By placing scripts at the bottom, the entire page can be loaded before any scripts are executed, ensuring that all style sheets and images referenced by the HTML content as well as the CSS can be queued before the browser pauses to execute scripts, thus increasing parallelism, and in effect client-side performance.

A side-effect of this change is that scripts that have been injected arbitrarily into the body can not depend on functionality provided by scripts which are loaded at the bottom. It may be possible to improve backwards compatibility in the future, but there are no specific plans to do so at this time. We recommend to do javascript bindings from the modules rather than inline.

Troubleshooting and tips

As the scripts are loaded at the bottom of HTML, make sure that all OutputPage-generated HTML tags are properly closed. If you are porting JavaScript code to ResourceLoader, make the default outer scope variable declarations explicitly use `window` as their parent, because the default scope in ResourceLoader is not global (e.g. not a window). Which means that code that previously became global by using `var something` in a `.js` file, is not local and is only global if you make it global by literally attaching it to the window object by using `window.something`.

An even better approach (instead of changing `var` to `window`, if needed) is to convert the existing extension's JavaScript code to an **object oriented structure**. (or, if the code is mainly about manipulating DOM elements, make a jQuery plugin)

Suppose your current structure looks like this:

```
var varname = ..;

function fn(a,b) {
}
```

In order to keep it working with existing implied globals, replace implied globals with real globals (which is should already be):

```
window.varname = ..;

window.fn = function( a, b ) {
}
```

To actually port it to an object oriented structure, you'd have something like this:

```
mw.fooBar = {
    varname: ...,
    fn: function( a, b ) {
    }
}
```

If you are using an external JavaScript library, which is primarily maintained outside of MediaWiki extension repository, patching as suggested above is undesirable. Instead, you should create an extra JavaScript file where the required variables and functions would be bound via the references to window. Then, when registering ResourceLoaderFileModule you should pass both scripts in one PHP array:

```
'scripts' => array( 'ext.myext.externalCode.js',
'ext.myext.externalCode.binding.js' )
```

This will weld the two scripts together before throwing them into a closure.

`ext.myext.externalCode.binding.js` should contain assignment statements like this:

```
window.libvar = libvar;
window.libfunc = libfunc;
```

References

- [1] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/Resources.php?view=markup>
- [2] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/OutputPage.php?view=markup>
- [3] <http://www.mediawiki.org/wiki/Special:Code/MediaWiki/77011>

ResourceLoader/JavaScript Deprecations

shortcut **RL/JD**

This page documents **deprecated identifiers that will be removed from mediawiki eventually** and there replacements as can be found in the "Future" columns below. All replacements and some new modules are available as-is since 1.17. Please use the modern replacements for new code and rewrite old code immediately. If you encounter no replacement and believe there should be one, please Let us know! Additionally, XML retrieved by invoking GET and POST methods on `index.php` is incompatible with HTML 5, which is the default as of 1.16 (but not on WMF yet^[1]). You should update code to use `api.php`, JSON format and `jQuery.ajax` immediately.

Please note the current implementation in svn^[2] may differ from the current release and what might eventually be included with mediawiki. Any code relying on details attained from svn may break.

Toggle grep script

There is a plain text list available of all identifiers with indicated replacements on this page that is suitable for automated searching of source code using Unix style commands such as:

```
grep -f deprecated.ids *.js
```

where `deprecated.ids` is a text file created from the list page. Make sure to not have a blank line at the end of the file. Note that due to some variables potentially being part of another (non-deprecated or custom) method, using 'grep' is not recommended for automated reviews (for example `addPortletLink` and `mw.util.addPortletLink`, and 'ta' and 'var = takeMyCustomVariable').

ajax.js ^[3]

Identifier	Type	Future
sajax_debug_mode	string	<p>» jQuery.ajax ^[4]</p> <p>Use AJAX ^[5] functions to access pages and the API directly.</p> <pre><code>\$('#mybox').load('/wiki/Template:ResourceLoader_navigation#bodyContent');</code></pre>
sajax_request_type	string	<pre><code>\$.get (mw.config.get('wgScript'), { title: 'ResourceLoader', action: 'raw' }, function(data) { /* Call was successful do something with "data" */ }); \$.getJSON(mw.util.wikiScript('api'), { format: 'json', action: 'query', titles: 'Main Page', prop: 'info' }, function(obj) { /* API call was successful do something with obj */ });</code></pre> <p>For more information see documentation for <code>.load()</code> ^[6], <code>.get()</code> ^[7], <code>.getJSON()</code> ^[8] and the other jQuery Ajax functions ^[5].</p> <p>See also this guide from WikiProject User scripts, for an example of how to edit a wiki page using AJAX.</p>
sajax_debug	function	
sajax_init_object	function	
wfSupportsAjax	function	
sajax_do_call	function	
		<pre><code>\$.get (mw.util.wikiScript(), { action: 'ajax', rs: 'SomeExtensionFunction', rsargs: [arg1, arg2, arg3] // becomes &rsargs[]=arg1&rsargs[]=arg2... });</code></pre> <p><i>Note: These calls using action=ajax should be rewritten to use an API module</i></p>

ajaxwatch.js

Identifier	Type	Future
wgAjaxWatch	object	mediawiki.action.watch.ajax.js ^[9] (No public identifiers)

block.js

Identifier	Type	Future
considerChangingExpiryFocus	function	mediawiki.special.block.js ^[10] (No public functions)
updateBlockOptions	function	

changepassword.js

Identifier	Type	Future
onNameChange	function	None/Removed
onNameChangeHook	function	

edit.js

The toolbar interfaces will likely migrate into an extension. The default bundle would include the Extension:WikiEditor, which already has an advanced API that replaces all of this.

Identifier	Type	Future
currentFocused	undefined	
addButton	function	mw.toolbar.addButton ^[11]
mwInsertEditButton	function	
mwSetupToolbar	function	
insertTags	function	mw.toolbar.insertTags ^[11]
scrollEditBox	function	
mwEditButtons	Array	mw.toolbar.buttons ^[11]

enhancedchanges.js

Identifier	Type	Future
toggleVisibility	function	None/Removed

history.js

Identifier	Type	Future
historyRadios	function	mediawiki.action.history.js ^[12] (No public functions)
diffcheck	function	
histrowinit	function	

htmlform.js

Identifier	Type	Future
htmlforms	object	None/Removed

IEFixes.js ^[13]

Identifier	Type	Future
doneIETransform	object	None of these should ever be used by anyone. :)
doneIEAlphaFix	object	
expandedURLs	object	
hookit	function	
fixalpha	function	
relativeforfloats	function	
setrelative	function	
onbeforeprint	function	
onafterprint	function	

metadata.js

Identifier	Type	Future
attachMetadataToggle	function	No public functions

mwsuggest.js

Identifier	Type	Future
------------	------	--------

os_map	object	<pre> » jQuery.autocomplete()^[14] \$('#mw-search').autocomplete({ minLength: 2, source: function(request, response) { \$.getJSON(mw.util.wikiScript('api'), { format: 'json', action: 'opensearch', search: request.term }, function(arr) { if (arr && arr.length > 1) { response(arr[1]); } else { response([]); } }); } }); </pre>
os_cache	object	
os_cur_keypressed	number	
os_keypressed_count	number	
os_timer	object	
os_mouse_pressed	boolean	
os_mouse_num	number	
os_mouse_moved	boolean	
os_search_timeout	number	
os_autoload_inputs	object	
os_autoload_forms	object	
os_is_stopped	boolean	
os_max_lines_per_suggest	number	
os_animation_steps	number	See documentation ^[15] for more information.
os_animation_min_step	number	
os_animation_delay	number	
os_container_max_width	number	
os_animation_timer	object	
os_use_datalist	boolean	
os_Timer	function	
os_Results	function	
os_AnimationTimer	function	
os_MWSuggestInit	function	
os_initHandlers	function	
os_hookEvent	function	
os_eventKeyup	function	
os_processKey	function	
os_eventKeypress	function	

os_eventKeydown	function
os_eventOnsubmit	function
os_hideResults	function
os_decodeValue	function
os_encodeQuery	function
os_updateResults	function
os_setupDatalist	function
os_getNamespaces	function
os_updateIfRelevant	function
os_delayedFetch	function
os_fetchResults	function
os_getTarget	function
os_isNumber	function
os_enableSuggestionsOn	function
os_disableSuggestionsOn	function
os_eventBlur	function
os_eventFocus	function
os_setupDiv	function
os_createResultTable	function
os_showResults	function
os_operaWidthFix	function
f_clientWidth	function
f_clientHeight	function
f_scrollLeft	function
f_scrollTop	function
f_filterResults	function
os_availableHeight	function
os_getElementPosition	function

os_createContainer	function
os_fitContainer	function
os_trimResultText	function
os_animateChangeWidth	function
os_changeHighlight	function
os_HighlightClass	function
os_updateSearchQuery	function
os_eventMouseover	function
os_getNumberSuffix	function
os_eventMousemove	function
os_eventMousedown	function
os_eventMouseup	function
os_createToggle	function
os_toggle	function

prefs.js

Identifier	Type	Future
tabbedprefs	function	mediawiki.special.preferences.js ^[16] (None public functions)
uncoversection	function	
checkTimezone	function	
timezoneSetup	function	
fetchTimezone	function	
guessTimezone	function	
updateTimezoneSelection	function	

preview.js^[17]

Identifier	Type	Future
doLivePreview	function	

protect.js^[18]

Identifier	Type	Future
ProtectionForm	object	

rightclickedit.js

Identifier	Type	Future
setupRightClickEdit	function	mediawiki.action.view.rightClickEdit.js ^[19] (No public functions)
addRightClickEditHandler	function	

search.js

Identifier	Type	Future
mwSearchHeaderClick	function	mediawiki.special.search.js ^[20] (No public functions)
mwToggleSearchCheckboxes	function	

upload.js^[21]

Identifier	Type	Future
wgUploadWarningObj	object	Extension:UploadWizard
wgUploadLicenseObj	object	
licenseSelectorCheck	function	
wgUploadSetup	function	
toggleUploadInputs	function	
fillDestFilename	function	
toggleFilenameFiller	function	

wikibits.js^[22]

Identifier	Type	Future
------------	------	--------

clientPC	string	» jQuery.client
is_gecko	boolean	
is_safari	boolean	
is_safari_win	boolean	
is_chrome	boolean	
is_chrome_mac	boolean	
is_ff2	boolean	
is_ff2_win	boolean	
is_ff2_x11	boolean	
webkit_match	object	» jQuery.support ^[23] maybe?
ff2_bugs	boolean	
ie6_bugs	boolean	

doneOnloadHook	boolean	<p>» jQuery</p> <p>Use</p> <pre> jQuery(document).ready(function(\$) { /* your inline code to be executed after the page is loaded */ });</pre>
onloadFuncs	object	<pre>// or: jQuery(document).ready(LoadMyApp); </pre>
addOnloadHook	function	<p>Or shorthand:</p> <pre> jQuery(function(\$) { /* your inline code to be executed after the page is loaded */ });</pre>
runOnloadHook	function	<pre>// or: \$(LoadMyApp); </pre>
killEvt	function	<p>» jQuery</p> <pre> \$(window).click(function(e) { e.preventDefault(); /* do something else */ }); </pre>
loadedScripts	object	<i>redundant</i>
importScript	function	» mediaWiki.loader
importScriptURI	function	
importStylesheet	function	
importStylesheetURI	function	
appendCSS	function	» mediaWiki.util.addCSS

addHandler	function	<p>» jQuery Binding</p> <p>Using <code>. bind()</code> ^[24] one add handlers. Some of the more common events have their own shortcuts (such as <code>. click()</code> ^[25].)</p> <pre> // Bind multiple events to an element and attach an anonymous function \$('#fooBar').bind('mouseenter mouseleave', function() { alert('#fooBar is moused!'); }); // Bind multiple events with different handlers to a single element \$('#fooBar').bind({ click: function() { // do something on click }, mouseenter: function() { // do something on mouseenter } });</pre>
addClickHandler	function	
removeHandler	function	
hookEvent	function	<pre>// Bind a handler to the click events of an earlier defined DOM element \$(myElement).click(myFunction); </pre> <p>For more information about this, and about unbinding and live-binding see: jQuery Docs on <code>unbind()</code> ^[26], <code>bind()</code> ^[24], <code>click()</code> ^[25] and <code>live()</code> ^[27]</p>
mwEditButtons	object	<i>deprecated</i>

mwCustomEditButtons	object	<p>» Vector / UsabilityInitiative</p> <p>The new toolbar in Vector (which also works fine in Monobook, see here ^[28]) has built-in function to customize the toolbar. For more info see Toolbar customization on the usabilitywiki. For examples to insert buttons using the native script see Toolbar customization/Library. For a script to insert buttons using a simple syntax similar to the old way check out [29][30]</p> <pre> importScriptURI('http://meta.wikimedia.org/w/index.php?title=User:Krinkle/insertVectorButtons.js' + '&action=raw&ctype=text/javascript'); function kCustomMainInsertButton_config() { // Welcome : kCustomMainInsertButton('welcome', //imageId 'http://commons.wikimedia.org/w/thumb.php?f=Nuvola_apps_edu_languages.svg&w=22', //imageFile 'Welcome', //speedTip '{{Welcome}}\~\~\~\~', //tagOpen '', //tagClose '' //sampleText); // Test2 : kCustomMainInsertButton('test2', //imageId 'http://commons.wikimedia.org/w/thumb.php?f=Nuvola_apps_important.svg&w=22', //imageFile 'Test2', //speedTip '{{subst:test2 ', //tagOpen '}}\~\~\~\~', //tagClose '' //sampleText); } </pre>
tooltipAccessKeyPrefix	string	mw.util.tooltipAccessKeyPrefix
tooltipAccessKeyRegexp	object	mw.util.tooltipAccessKeyRegexp
updateTooltipAccessKeys	function	mw.util.updateTooltipAccessKeys
ta	object	<i>deprecated</i>
akeytt	function	<i>deprecated</i>

checkboxes	object	» jQuery.checkboxShiftClick
lastCheckbox	object	
setupCheckboxShiftClick	function	
addCheckboxClickHandlers	function	
checkboxClickHandler	function	
showTocToggle	function	
toggleToc	function	
ts_image_path	string	» jQuery.tablesorter
ts_image_up	string	
ts_image_down	string	
ts_image_none	string	
ts_europeandate	boolean	
ts_alternate_row_colors	boolean	
ts_number_transform_table	object	
ts_number_regex	object	
sortables_init	function	

ts_makeSortable	function	
ts_getInnerText	function	
ts_resortTable	function	
ts_initTransformTable	function	
ts_toLowerCase	function	
ts_dateToSortKey	function	
ts_parseFloat	function	
ts_currencyToSortKey	function	
ts_sort_generic	function	
ts_alterate	function	
changeText	function	<p>» jQuery</p> <p>Using <code>.text()</code> one can get or set the text:</p> <pre> // Set text of an earlier defined element \$(myElement).text('New text!');</pre>
getInnerText	function	<pre>// Set text for multiple elements at once \$('.mw-userlink').text('Mr. Foobar'); // Get the text of an element \$(myElement).text(); </pre>
escapeQuotes	function	
escapeQuotesHTML	function	» mediaWiki.html
addPortletLink	function	» mediaWiki.util.addPortletLink

jsMsg	function	» mediaWiki.util.jsMessage
injectSpinner	function	
removeSpinner	function	
getElementsByClassName	function	<p>» jQuery</p> <p>Use the CSS Selector in jQuery to select elements by classname. Here examples for the three most common usecases of <code>getElementsByClassName()</code>:</p> <pre> // Get everything with a classname \$('.myclass'); // Get all of element X (ie. table) with a classname \$('table.wikitable'); // Get all of something within another element \$(myDomElement).find('.myclass'); \$(myDomElement).find('> .myclass'); // only direct children // Get all class-elements that are direct children of an element \$('#someElement .myclass'); \$('#someElement > .myclass'); // only direct children </pre>
redirectToFragment	function	

Documentation	Migration guide (users) · Migration guide (developers) · Developing with ResourceLoader · Default modules · Vocabulary
Project information	Status updates · Version 1 Design Specification (tasks) · Version 2 Design Specification (tasks) · Requirements
Other	JavaScript Deprecations

References

- [1] https://bugzilla.wikimedia.org/show_bug.cgi?id=27478
- [2] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/>
- [3] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/ajax.js?view=markup>
- [4] <http://api.jquery.com/jQuery.ajax/>
- [5] <http://api.jquery.com/category/ajax/>
- [6] <http://api.jquery.com/load/>
- [7] <http://api.jquery.com/jQuery.get/>
- [8] <http://api.jquery.com/jQuery.getJSON/>
- [9] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.action/mediawiki.action.watch.ajax.js?view=markup>
- [10] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.special/mediawiki.special.block.js?view=markup>
- [11] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.action/mediawiki.action.edit.js?view=markup>
- [12] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.action/mediawiki.action.history.js?view=markup>
- [13] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/IEFixes.js?view=markup>
- [14] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/jquery.ui/jquery.ui.autocomplete.js?view=markup>
- [15] <http://jqueryui.com/demos/autocomplete/>

- [16] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.special/mediawiki.special.preferences.js?view=markup>
 - [17] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/preview.js?view=markup>
 - [18] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/protect.js?view=markup>
 - [19] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.action/mediawiki.action.view.rightClickEdit.js?view=markup>
 - [20] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/resources/mediawiki.special/mediawiki.special.search.js?view=markup>
 - [21] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/upload.js?view=markup>
 - [22] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/skins/common/wikibits.js?view=markup>
 - [23] <http://api.jquery.com/jQuery.support/>
 - [24] <http://api.jquery.com/bind/>
 - [25] <http://api.jquery.com/click/>
 - [26] <http://api.jquery.com/unbind/>
 - [27] <http://api.jquery.com/live/>
 - [28] <http://usability.wikimedia.org/w/index.php?title=Wikipedia:Sandbox&action=edit&useskin=monobook>
 - [29] http://usability.wikimedia.org/wiki/Talk:Toolbar_customization#Switching_from_Monobook
 - [30] <http://meta.wikimedia.org/wiki/User:Krinkle/Scripts/insertVectorButtons#Description>
-

Testing

Testing portal

MediaWiki supplements manual testing/bug reporting ^[1] with automated testing. Here's a summary of current automated testing and request for next steps, as of 5 July 2011. This portal will point you to resources so you can run test suites, write tests, and improve the testing infrastructure.

Testing approaches and how we do them

There are several overlapping approaches in testing:

- fuzz testing: you inject random values and look at the results. Developers, testers, and users do this manually.
- unit testing ^[2]: making sure your API public methods works as expected. Automated via PHPUnit, QUnit, and Jenkins.
- integration testing: testing the integrated whole as it operates together; looking at an installation and trying to break it and filing bug reports; manually or via automation
- behaviour testing : trying to reproduce an user scenario (as you can program into Selenium)
- logical testing : just invert any logical conditions (aka: == is made != while != are made ==).
- code coverage : in a method, try to inject values to test all possible code paths and conditions (usually automated, in our case via parsertests & Jenkins)

Test suites

MediaWiki internal

- Parser tests
 - Currently automatically run on trunk and integrated into CodeReview overview
 - Hooks for extensions to add tests (not yet automated)
 - Cite (ref)
 - LabeledSectionTransclusion
 - ParserFunctions
 - Poem
 - SlippyMap
 - t/ unit tests
 - not totally functional bugzilla:20112
 - Some of the more generic ones (eol checks, bom checks) moved to tools/code-utils in r54922
 - tests/ unit tests
 - checkSyntax
 - In maintenance/
 - Needs upload capability
-

External suites

Extension test framework

Client-side test suites

- There's some stuff going on with some of the UsabilityInitiative extension, not yet automated.
- Some talk of doing Selenium-based browser-hosted testing but this has not yet been implemented.
 - Some Selenium tests
 - Selenium no longer the way forward Sumanah 19:15, 1 July 2011 (UTC)
- Using HTML rendering engine (Gecko? WebKit?) to run user interface tests
- some JS testing notes ^[3]
 - Once you get QUnit tests, have a look at TestSwarm ^[4], a distributed JS tester.

Performance testing

It would likely be wise to set up some automated performance testing; e.g., checking how much time, how many DB queries, how much memory used, etc. during various operations (maybe the same tests running above).

While performance indicators can vary based on environment and non-deterministic factors, logging and graphing the results of multiple iterations from a consistent testing environment can help us with two important things:

- Identify performance regressions when we see an unexpected shift in the figures
- Confirm performance improvements when we make some optimization

See for instance Mozilla's performance test graphs for Firefox:

- <http://graphs.mozilla.org/dashboard/?tree=Firefox>

(more FF stuff via <https://developer.mozilla.org/en/Tinderbox>)

How to report a bug

https://www.mediawiki.org/wiki/How_to_report_a_bug

Other notes

- Extension Testing
 - Mobile browser testing
 - Database testing
 - Continuous integration server
 - Manual:Unit testing
 - Fixing broken tests
-

Links

- Search for testing ^[5] in MediaWiki.org.
- Test wiki: <http://test2.wikipedia.org>

References

- [1] <https://bugzilla.wikimedia.org/>
- [2] http://diveintopython.org/unit_testing/diving_in.html
- [3] <http://sayspy.blogspot.com/2009/08/testing-javascript-code-and-releasing.html>
- [4] <http://testswarm.com/>
- [5] <http://www.mediawiki.org/w/index.php?title=Special%3ASearch&search=testing&fulltext=Search>

Writing testable PHP code

Some notes on writing testable, code, to be expanded on at some point.

Don't assume global context

Accessing global variables (e.g. `$wgRequest`) without declaring them first with the `global` keyword will cause failures and `E_NOTICE` messages to be generated if they are accessed in a non-global context.

Don't create new global variables

While putting information in global variables seems easy, it makes the code less predictable. By relying on global variables, you are making it difficult to isolate functionality. A singleton class is better for testing (but, still, less than ideal).

Rely only on direct inputs

While this is not always achievable, it is best to write code that depends only on direct inputs. That is, a class only uses the information it is passed and does not rely on singletons or globals to get “out-of-band” information.

Do not use `exit ()`

Exiting from a script abruptly should almost never be done. Doing so will make your code untestable by PHPUnit. If your code encounters an unexpected error, the proper thing to do is throw an exception like:

```
throw new MWException( "Oh noes!" );
```

This will allow PHPUnit and MediaWiki to exit properly and provide informative information such as stack traces to developers.

```
Time: 10 seconds, Memory: 97.25Mb

There were 35 incomplete tests:
[snip]

OK, but incomplete or skipped tests!
Tests: 821, Assertions: 78982, Incomplete: 35, Skipped: 1.
```

Running an individual test set:

```
$ cd tests/phpunit
$ php phpunit.php includes/IPTest.php

PHPUnit 3.5.13 by Sebastian Bergmann.

.....

Time: 1 second, Memory: 14.75Mb

OK (29 tests, 6025 assertions)
```

Writing Unit Test for Extensions

The recommended bootstrapping method for extensions is to use the `UnitTestsList` hook. By registering your tests with this hook you can use the mediaWiki core bootstrap support for things like autoloading classes, global functions etc. Unless your code is truly a standalone library, it's **NOT** recommended to include a few random MediaWiki core files with custom bootstrapping and attempt to make your tests stand alone.

To exclusively run your extensions tests, duplicate the `suite.xml` file in `tests/phpunit/suite.xml` to `suite.extensions.xml` only include the "extensions" testsuite as a child of testsuites. Then run the test with the command: `php phpunit.php --configuration extensions.suite.xml`

To test a single file you can pass the test file as a parameter to `php phpunit.php`

Code

Let's assume our extension is named Fruits. In `Fruits/Fruits.php` main file add:

```
$wgHooks['UnitTestsList'][] = 'efFruitsRegisterUnitTests';
function efFruitsRegisterUnitTests( &$files ) {
    $testDir = dirname( __FILE__ ) . '/';
    $files[] = $testDir . 'AppleTest.php';
    return true;
}
```

In `AppleTest.php` add:

```
<?php
class AppleTest extends MediaWikiTestCase {
    protected function setUp() {
        parent::setUp();
        $this->apple = Fruit::factory( 'apple' );
    }
}
```

```

    }
    protected function tearDown() {
        unset( $this->apple );
        parent::tearDown();
    }
    public function testIsEdible() {
        $this->assertTrue( $this->apple->isEdible(), "apples
should be edible." );
    }
}

```

To make easier to run test, you could create a Makefile something like this:

```

tests:
    ls *Test.php | xargs -n1 php ../../../../tests/phpunit/phpunit.php

```

Writing unit tests

The PHPUnit Manual ^[10] provides good instructions for understanding and developing unit tests. Pay particularly close attention to the sections on writing ^[11] and organizing ^[12] tests.

Developers new to unit testing in MediaWiki should use SampleTest.php ^[13] as a starting point – it contains helpful comments that will ease the process of learning to write unit tests.

Another resource is the slides ^[14] from the PHPUnit Best Practices ^[15] talk that Sebastian Bergmann gave at OSCON 2010.

Write Testable Code

Please try to write testable code.

MediaWiki was not written with the objective of being testable. It uses global variables all over the place and static methods in many places. This is a legacy that we have to accept, but try not to introduce these things into new code, and try to change going forward.

A good resource might be Miško Hevery's Guide to Testability ^[16]. (Miško Hevery ^[17] is [one of?] Google's Agile Coaches.)

Telling a story

Test output should tell a story (output options).

Number of Assertions

Only one assertion per test unless there is a good reason (expensive tests may need to be grouped).

Grouping tests

PHPUnit allows tests to be put into arbitrary groups. Groups of tests can be selected for execution or excluded from execution when the test suite is run (see the @group annotation ^[18], The Command-Line Test Runner ^[19] and XML Configuration File ^[20] documentation in the PHPUnit manual for additional details.)

To add a test (or class) to a group, use the @group annotation in the docblock preceding the code. For example:

```
/**
 * @group Broken
 */
class HttpTest extends PHPUnit_Framework_TestCase {
    ...
}
```

Eight groups are currently used in the MediaWiki unit tests:

- **Broken:** Put broken tests into group Broken. Tests in this group will not be run (as is configured in `tests/phpunit/suite.xml`^[21]).
- **Database:** Tests that require database connectivity should be put into group Database.
- **Destructive:** Tests that alter or destroy data should be put into group Destructive.
- **Fundraising:** Tests related to WMF fundraising (CentralNotice, DonationInterface, etc.)
- **Search:** Tests that use MediaWiki's built-in search put into group Search.
- **SeleniumFramework:** Tests that require SeleniumFramework to be installed should be put in group SeleniumFramework.
- **Stub:** Put test stubs into group Stub. Tests in this group will not be run (as is configured in `tests/phpunit/suite.xml`^[21]).
- **sqlite:** Tests that use SQLite should be put into group sqlite.
- **Upload:** Tests that upload files should be put into group Upload.
- **Utility:** Currently unused by any test. Tests in this group will be not be run (as is configured in `tests/phpunit/suite.xml`^[21]).

To test only a particular group, use the `--group` flag from the command line:

```
php phpunit.php --group Search
```

Developing best practices

Developers should avoid inventing new conventions or borrowing conventions from other frameworks; using the already well-established PHPUnit conventions will serve to make MediaWiki's tests both useful and usable. Pay particularly close attention to the sections in the PHPUnit manual on writing^[11] and organizing^[12] tests.

How to help

- Make sure the tests run on your system -- if they don't, either something's broken or the tests are making bad assumptions, and it needs fixing either way!
- Find and fix any testing problems on non-MySQL database backends:
 - (SQLite is used for test runs on <http://integration.mediawiki.org/ci/>)
 - /PostgreSQL
 - /Oracle
 - /MS SQL Server
- Find code that needs testing and write tests

Notes

- [1] <http://www.phpunit.de/>
- [2] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/phpunit>
- [3] In MediaWiki v1.17 and earlier, unit tests were located in the `maintenance/tests/phpunit` directory.
- [4] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/IP.php?view=markup>
- [5] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/phpunit/includes/IPTest.php?view=markup>
- [6] <http://pear.php.net>
- [7] <http://pastebin.com/iP1rgc8Z>
- [8] <http://stackoverflow.com/questions/1528717/phpunit-require-once-error>
- [9] <http://www.phpunit.de/manual/current/en/installation.html>
- [10] <http://www.phpunit.de/manual/current/en/automating-tests.html>
- [11] <http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html>
- [12] <http://www.phpunit.de/manual/current/en/organizing-tests.html>
- [13] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/phpunit/includes/SampleTest.php?view=markup>
- [14] <http://assets.en.oreilly.com/1/event/45/PHPUnit%20Best%20Practices%20Presentation.pdf>
- [15] <http://www.oscon.com/oscon2010/public/schedule/detail/12510>
- [16] <http://misko.hevery.com/2009/03/09/guide-to-testability-is-now-downloadable/>
- [17] <http://misko.hevery.com/about/>
- [18] <http://www.phpunit.de/manual/current/en/appendixes.annotations.html#appendixes.annotations.group>
- [19] <http://www.phpunit.de/manual/current/en/textui.html>
- [20] <http://www.phpunit.de/manual/current/en/appendixes.configuration.html>
- [21] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/phpunit/suite.xml>

JavaScript unit testing

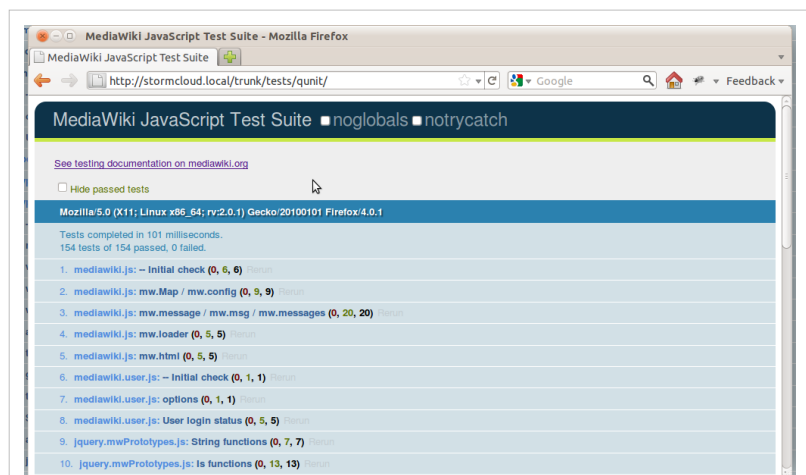
Unit testing in MediaWiki for its JavaScript code base is performed using the QUnit JavaScript Testing framework.^[1]

The unit tests are located in the `tests/qunit`^[2] directory^[3]. Tests are organized into a directory structure that matches the directory structure of the code that they are testing. For example: The unit tests for `resources/mediawiki.util/mediawiki.util.js` can be found in `tests/qunit/suites/resources/mediawiki.util/mediawiki.util.test.js`.

Running the unit tests

Run the unit tests from the browser.

1. Browse to `http://location.of/wiki/tests/qunit`.
2. There's no step 2, sit back and wait.
3. Nope, no step 3 either.



If all goes well, the tests will quickly run and show a green bar at the top like this.

Completeness test

In order to enable completeness test plugin, set "completeness=1" in the query string

- Browse to `http://location.of/wiki/tests/qunit/?completeness=1`.

How to help?

If you're not a developer or don't feel comfortable enough to write a unit test, you can still be of great help by running unit tests:

- Make it a habit to run unit tests after updating your subversion checkout. Any problems? See if you can find a cause and let the committer know by posting a comment under that revision (mark it `FIXME` as well).
- Join Wikimedia's TestSwarm and let it run in the background (TestSwarm is not configured yet)

Another way to help out is by expanding our unit test coverage of the MediaWiki JavaScript library. Run the CompletenessTest and find out which methods don't have unit tests yet and write one.



How to write a QUnit test in MediaWiki

Todo

- The file should be named after the module (or file) it is testing.
- Inside the test suite file should be one, and one exactly, call to QUnit's `module()`. The string passed to `module()` must match the filename (but without the `.test.js` part). This is to enable sectioning in the QUnit output as well as for TestSwarm to allow distributed and asynchronous testing of each module separately.

TestSwarm

Todo: Branch off to a separate wiki page

TestSwarm is a system to parcel out JavaScript-based tests to multiple browsers. As many folks can connect as desired, and tests will be automatically sent to whatever connected browser and recorded based on success and browser version etc. Because MediaWiki jobs submitted to TestSwarm are assigned to all the browsers we support at that time (even more rarely-used browsers, regardless of whether these are connected to the swarm when the commit is made), we can ensure that test coverage gets run on all of them, not just whatever a developer has handy when they make a commit.

- On Toolserver: `http://toolserver.org/~krinkle/testswarm/`
 - MediaWiki commits to core modules and test suites^[4] (previously, pre-r88431: here^[5]).
- From conversation with Krinkle: "TestSwarm does this: It automatically checks out every new revision (related to js-resources) from SVN and then it sends them to different browsers to run the test and summarizes the results – in order to easily track if something brakes, and when it does we can see the exact revision in which the breakage started (even if new commits were made), and where (modulename and testfunction) of the breakage."

- In a sentence, relation between QUnit and TestSwarm is:

```
"Automated distributed continuous unit testing for JavaScript"
(where QUnit="unit testing", and TestSwarm="Automated distribution, continuously.")
```

- It's a bit like CruiseControl for PHP Unit ("Automated continuous integration for PHP"), except for the "distribution" part. Which CruiseControl doesn't do, but also doesn't have to do, because:
 - Every browser is different in terms of how far and which version of the ECMAScript-specification was implemented (ie. JavaScript).

However PHP is pretty much static/similar on all platforms. Which is why running PHPUnit on your localhost is enough to know that the code is good if (it passes the test). But due to cross-browser differences, running QUnit on your localhost in Firefox doesn't say all that much. That's why we need to run tests in all browsers we support (IE7,8,9, Firefox 3,4,5 etc.).

- TestSwarm keeps track of which module in which revision is tested each of these browsers. And whenever a client connects TestSwarm sends tests that haven't been run in that svn-revision / browser-name-version combination yet.

<-- TODO we might want to use a drawing to explain it all -->

With TestSwarm we crowdsource the "grid". And aside from the advantage in maintenance (all clients being crowdsourced), TestSwarm generally does a better job at presenting its information about the status of the unit tests and has the ability to proactively correct bad results coming in from clients (As any web developer knows: Browsers are surprisingly unreliable (inconsistent results, browser bugs, network issues, etc.). Another great aspect of the swarm is that people can opt in with their browsers at any time as the swarm keeps all old revisions. So if a certain type of browser hasn't been in the swarm it can catch up at any time. Or a project manager could reset an old revision to 'untested' and have the swarm redistribute as if it were a new one.

More about TestSwarm functioning and comparison to other platforms ^[6]

jQuery's manual describes this setup with TestSwarm and QUnit as "Automated distributed continuous integration for JavaScript" (where QUnit is "unit testing", and TestSwarm "Automated distribution, continuously.).

Developers write unit tests for their modules with QUnit. Both the modules and the test suites are in Subversion. Developers can also run the test suite locally (/phase3/tests/qunit/index.html).

TestSwarm automatically checks out every new revision (a job) and creates to-do runs (a run is a combination between a browser-engine version, a MediaWiki module and subversion revision). When a client connects, the swarm sends out "runs" to run that haven't been ran yet and the client pushes back the results.

Current Problems

- ~~The test runner page has no pointer to this documentation~~ (added r88734; not pretty but at least it's in)
- Test files often have the same name as the files they are testing, eg 'mediawiki.js' or 'jquery.colorUtil.js'. This makes code maintenance more difficult because bare filenames are often shown when navigating or working with code. Consider using the word 'test' visibly in the filename.
 - For instance, our phpunit test case files are almost always named by appending 'Test' to the tested class/file's name:
 - Block.php <-> BlockTest.php
 - IP.php <-> IPTest.php
 - Title.php <-> TitleTest.php
 - LanguageConverter.php <-> LanguageConverterTest.php
- It looks like new test files must be manually added to index.html. Among other things this static list means that extensions cannot add test cases except by implementing a second test runner page?

- *Krinkle's got some ideas for this, but for now we're concentrating on the core tests which are all bundled together, so easy enough to work with in the meantime. For more info/discussion see this thread.*

Notes

- [1] <http://docs.jquery.com/QUnit>
- [2] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/qunit>
- [3] Before r88431 , unit tests were located in the `resources/test` directory.
- [4] <http://toolservr.org/~krinkle/testswarm/user/MediaWiki/>
- [5] <http://toolservr.org/~krinkle/testswarm/user/KrinkleBot/>
- [6] <https://github.com/jquery/testswarm/wiki>

Parser tests

Parser tests are snippets of wikitext that get run through the parser. The actual output is compared with the desired result, and thus, parser tests can be helpful at spotting regressions. They reside in `phase3/tests/parserTests/parserTests.txt`

Syntax & execution

Syntax is as following:

```
!! test
Simple paragraph
!! input
This is a simple paragraph.
!! result
<p>This is a simple paragraph.
</p>
!! end
```

To run the parser tests, go to the root directory of your local MediaWiki installation and execute the following from the command line:

```
php tests/parserTests.php
```

Style guide

Style guide

Brandon Harris is currently working on a MediaWiki interface style guide.

The style guide has been broken down into sections.

- Color selection and accessibility — General information about using color and designing for accessibility
- Forms — General information about standard form elements and their behavior
- Error handling and validation — Information about input validation and error handling
- Specialized form elements — Specialized form elements and ilk (e.g., data thermometers, star bars, call-to-action buttons)
- Assets and gallery — Overview of assets and images used in this documentation

Color selection and accessibility

This document is a work in progress.

This document is part of the MediaWiki style guide. This section concerns itself with color choices and elements of user accessibility that should be taken into account when designing systems.

Color Selection

Color selection within user-interfaces can be tricky. It is helpful to understand how human beings react to various colors and the psychological reasons behind these reactions.

Color can be divided into two broad categories: "warm" and "cool". As a rule of thumb, *warm* colors are those with lots of red and yellow in their makeup; *cool* colors are those with *any* blue. Browns are warm; greys and blacks are perceived as "cool".

Warm:	Red	Orange	Yellow	Brown
Cool:	Blue	Green	Purple	Grey

The color *red* requires special consideration. No other color demands attention as much as red because human beings are *biologically wired* to respond to the color red as a stressor. There are measurable, biological effects to its presence (such as increased heart rate). Quite simply, this is because of the color of human blood. We know, innately, that "red" equals "bad" and demands immediate attention.

Likewise, *Green* is perceived as being positive. Green is the color of life (trees, grass, etc.). Human culture has elevated the color green to one meaning "forwardness" and "safety."

Yellow is a difficult color to work with in computers. Computer monitors (and television sets) do not generate the color yellow because pixels are comprised of only three colors: red, green, and blue. Any yellow color is technically generated as a low-intensity brown (with green, red, and white pixels firing in a moire pattern).

Research References

- Color Psychology in Web Design - Big Websites Case Study ^[1]
 - Empathizing Color Psychology in Web Design ^[2]
-

- Color Psychology and Marketing ^[3]
- Psychology of Colors: Web Design Colors ^[4]
- Color Psychology for Webpages ^[5]

Note: much of the color research found is duplicative; this is because color research is a known, well-explored area of study.

Alert Color versus Chrome Color

Since different sites have different color schemes, colors that attract attention can be different. Accordingly, we will refer to color types in the following manner:

- *Chrome color* is color that falls to the background of the user's perception. Chrome color is seen and understood to be part of the site's "operating system" and does not attract attention.
- *Alert color* is color that rises to the front of the user's perception. Alert color invites attention and exploration.

Most MediaWiki installs that use the Monobook or Vector skins will have a primary "chrome" color scheme of greys and blues. In this case, "alert" colors will have to be warm. In contrast, skins that are warmer in nature will use cooler colors for alerts.

Accessibility for Visually Impaired Users

There are many things to think about when designing forms that can be used effectively by users with various types of visual impairment.

Color blindness

There are several forms of color-blindness and they have subtle differences. This can be difficult to design for.

In general, avoiding the use of color as the *only* indicator of something in a form is best practice. Varying color intensity and lightness is key when designing indicators.

Poor eyesight

Subtle visual elements are often lost on users with poor eyesight. If something is important, it should be visually elevated. Elements of 5 pixels in size are usually too small to be easily distinguished; use 16 or more pixels in such cases.

Color intensity contrast is important in this regard. Two opposite colors can appear the same if they have the same intensity (amount of white).

Dyslexia

While not technically a vision impairment, certain design choices can wreck havoc to individuals with dyslexia.

Dyslexic individuals can have difficulty reading text on colored backgrounds when the intensity of the text color approaches that of the background.

Blocks of text should *never* be centered or justified. This creates a "gutter" effect which can appear as a moire pattern. Always use aligned text (left for left-to-right languages; right-aligned for right-to-left languages).

Other things to avoid include the use of capitals for emphasis, italics, and serif fonts.

Research References

- Lighthouse International: Effective Color Contrast ^[6]
 - Lighthouse International: Making Text Legible ^[7]
 - Web Design for Dyslexic Users ^[8]
 - Web Designing for Dyslexia ^[9]
-

- American Foundation for the Blind: Web Accessibility ^[10]

References

- [1] <http://www.pixel77.com/color-psychology-web-design-color-schemes-big-websites/>
- [2] <http://www.1stwebdesigner.com/design/color-psychology-website-design/>
- [3] <http://www.precisionintermedia.com/color.html>
- [4] http://www.pepfx.com/articles/web_design/webdesign_colors.php
- [5] <http://www.webknowhow.net/dir/Design/Miscellaneous/061016WebsiteColours.html>
- [6] <http://www.lighthouse.org/accessibility/design/accessible-print-design/effective-color-contrast>
- [7] <http://www.lighthouse.org/accessibility/design/accessible-print-design/making-text-legible/>
- [8] <http://www.dyslexia.com/library/webdesign.htm>
- [9] <http://www.hobo-web.co.uk/web-designing-for-dyslexic/>
- [10] <http://www.afb.org/Section.asp?SectionID=57&TopicID=167>

Style guide/Forms

This document is a work in progress.


This document serves as a form style guide for MediaWiki developers. It is intended to encourage development of a standard user experience for forms.

One of the goals of this document is to engender positive experiences for users. Forms need not be listless or boring; rather, even the most minor of forms should be energetic and interesting.


This document is focused on user-interactive forms. It does not cover style or best practices related to other elements of the user experience (such as tabulated data, or interactive lists).

Create Account


☒ **Username:**
Please view the [User Name Policy](#).


Unavailable
This username has been taken

☒ **Password:**
View our [Password Guidelines](#).

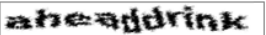
Strength: Okay


☒ **Confirm Password:**
Please re-enter your password.


Problem
The passwords entered do not match.

☒ **Email Address:** (Optional)
Learn why you should [Provide an Email Address](#).

☒ **Prove you're human:**
Enter the words you see below in the box without any spaces ([Why?](#))
Unable to see the image? Please [request that an account be created for you](#).



☒ Remember me (up to 30 days)

Web sites that accept postings from the public, like this wiki, are often abused by spammers who use automated tools to post their links to many sites.

This "CAPTCHA" helps us protect the wiki from spammers.

An example form, showing an account creation screen

An example form, showing a login screen

Field Elements

A single form can consist of one or more visible fields (there may be non-visible fields, such as hidden submission buttons for single field forms like a search box). Each field can have additional sub-elements, and their presence or absence is often form-specific.

Common elements include:

- **Label** - The element label. This is nearly *always* required.
- **Input Mechanism** - This is the field's input. This can be a text field, checkbox, select list, etc.
- **Required or Optional Notifier** - This indicates an element's required status
- **Error Notification** - This is an indicator to the user that there is a problem with the value placed within the element.
- **Help Trigger** - This is an icon or other actionable element that indicates to the user that additional information about the field is available.
- **Instructions** - This is typically a short sentence or two explaining what the field does and any parameters required.
- **Hint** - This is typically example text designed to help the user understand the type of data the element takes. Hints differ from instructions in that they are usually placed within the input mechanism.

A Note on Color Selection

See Style guide/Color selection and accessibility#Color Selection.

Label:

An standard form element

Label:

- ☐ Option 1
- ☐ Option 2
- ☒ Option 3
- ☒ Option 4

A grouping of checkboxes

Label:

- ☒ Option 1
- ☐ Option 2
- ☐ Option 3
- ☐ Option 4

A grouping of radio buttons

Accessibility for Visually Impaired Users

See [Style guide/Color selection and accessibility#Accessibility for Visually Impaired Users](#)

General Patterns

Field Layout

Forms should be laid out vertically in a single column. Human beings understand progress in forms as being vertically driven. Multiple columns are discouraged.

This is not a hard-and-fast rule, however; in some cases multiple columns can be utilized but this should be rare and typically only applied to a single form element grouping (such as "Password" and "Confirm Password").

Label Position

Generally, field labels should be placed above the field that they are referring to. The field element (input, select box, checkbox grouping, etc.) should be slightly indented below the label. Sufficient white space shall be applied beneath the element grouping to distinguish it from other elements on the page.

Labels are placed above for the following reasons:

- **Localization.** Translated labels may be significantly longer in other languages. Further, left-to-right languages (e.g., Arabic) naturally work better this way.
- **Human Eye Behavior.** Human beings naturally group elements in vertical proximity together. Continual Left-to-Right eye tracking is stressful.

An exception to this lies within single-element checkboxes, which case the label is to be placed on the same line as the checkbox, and immediately to the *right*.

Label:

Lorem Ipsum Dolor Sit

A select pull-down

☐ I accept the [terms of service](#).

A standard form element

Research References

- UX Movement: Users fill forms faster with top-aligned labels ^[1]
- UX Matters: Label Placement in Forms ^[2] (contains excessive eye-tracking data)
- [Luke Wroblewski Top, Right or Left Aligned Form Labels ^[3]
- CSS Tricks: Label Placement on Forms ^[4]
- Stack Exchange ^[5] (Good summary of Luke Wroblewski's research)

Checkboxes vs. Multi-Select

The use of *select* elements with the *multi* attribute is discouraged. In cases where the user can select multiple options from a discrete set, a *checkbox grouping* is suggested instead.

The drawbacks of the "multi select" are many but two primary reasons stand out:

1. The element is rarely used. As a result, not many users innately understand how they work.
2. Not all values are typically visible to the user. Additional scrolling is required. Again, users may not understand the control.

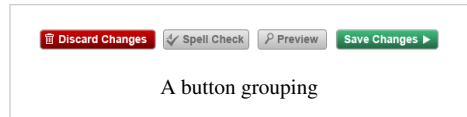
Buttons

For short forms with only one button, it is advised to place the button directly below the last field element and *left* aligned. Human eye tracking runs vertically and placing the button at the bottom is most natural psychologically.

However, for forms with multiple buttons, the buttons should be grouped together and right aligned at the bottom. This creates a psychological "break" in the flow and causes the user's short term memory to reset, allowing them to process the activity they are undertaking.

When there are multiple buttons, the order of the buttons should always follow the following pattern:

[destructive event] [reset event] [neutral event] [continuation event]



- A *destructive* event is one that will cancel the activity. All changes are lost. Destructive buttons should be colored red. Note that "back" buttons within wizards are *not* destructive; they are *reset* or *neutral*.
- A *reset* event is one where the form is returned to the default state. All changes are lost. Reset buttons should be colored grey or neutral. **The use of reset buttons is *strongly discouraged*.** There should be a significantly strong use case for them to ever appear.
- A *neutral* event is one where work will not be lost but the form will not be completed. Example: running a spell check, or in-page previews. Neutral buttons should be colored grey or neutral.
- A *continuation* event is moving to the next screen. This is typically a submission action but may be moving to the next page in a wizard. Continuation buttons should be colored green.

Text should rarely, if ever, be used for button actions. It is tempting to display "cancel" events as text but this should be avoided where possible; users have been trained to see buttons as actionable events and text links as data events.

Disabled Buttons

If a form should not be submitted until certain criteria are met (e.g., filled out completely), the continuation (submit) button should be disabled. Upon readiness, the button should become active and change color to indicate readiness.



Button Labels

In general, buttons should say *exactly* what they are doing and *always* re-stress what is happening. "Submit" is bad; "Save Changes" is better. "Cancel" is bad; "Discard Changes" is better.

Button Iconography

Button iconography is not required but is always a nice touch.

With the exception of continuation buttons, iconography should be placed to the left of the button text. Continuation button iconography should be placed to the right.

In general, icons on continuation buttons should be arrows pointing right. Buttons that take the user backwards in a series of steps should be arrows pointing left. Buttons that are purely destructive should have trashcan icons. Other iconography should dependant upon the function.

It is acceptable to only include iconography on continuation buttons.

Required and Optional Fields

In the past, common practice has been to indicate on a form when fields are *required* and assume all non-required fields are *optional*. However, over the past 20 years, users have been educated to assume that most fields are required and that few are optional (since this is generally true).

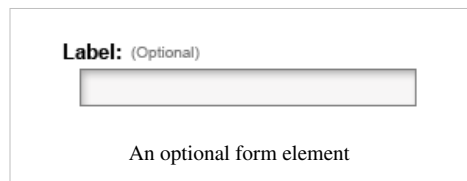
Indicating both optional *and* required fields is overkill, so only the *least common* type is to be indicated. Thus, if a form has 6 fields and only 2 are optional, the two optional entries should be indicated. If a form has only one required field, that should be called out.

For very short forms (such as a login form), it is not necessary to place any indications.

For long forms (more than 5 fields), a short notice should be given at the top indicating the general state (e.g., "All field are required", or "All fields are optional except where noted", etc.).

Display

Optional fields should be indicated with a text string, "(Optional)", directly after the field name. This should be in a smaller font and in a lower-intensity color.

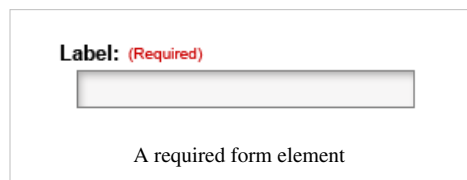


The diagram shows a form element with the label "Label: (Optional)" in a smaller, lighter font. Below the label is a rectangular input field. The entire element is enclosed in a thin gray border.

An optional form element

Required fields should be indicated with a text string, "(Required)", directly after the field name. This should be in a smaller font and a red color.

Required fields should have no other identifying marks (such as stars or icons) as this clutters the display and can be mis-interpreted as an error condition mark.

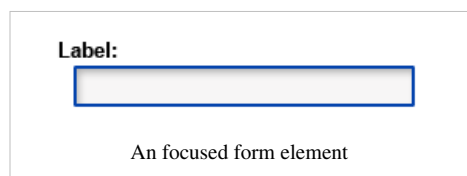


The diagram shows a form element with the label "Label: (Required)" in a smaller, red font. Below the label is a rectangular input field. The entire element is enclosed in a thin gray border.

A required form element

Focus Behaviors

When a text element is focused (has the cursor in it), its border should change color to indicate that it is active. Many modern browsers do this automatically.



The diagram shows a form element with the label "Label:" in a standard font. Below the label is a rectangular input field with a blue border, indicating it is focused. The entire element is enclosed in a thin gray border.

An focused form element

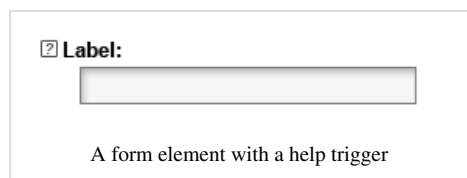
Help Triggers

Help triggers are intended to be in-form as tooltips and not designed to bring the user to another screen or open windows. Help triggers require the use of Javascript.

A help icon is placed immediately to the left of the element label. This is a small question mark icon. Hovering over it will cause a tooltip to appear with additional information.

In forms where the majority of elements have a help trigger, it is best practice to include one for *every* field for completeness' sake.

When a help trigger is present, the element's label should *also* trigger the tooltip. Help icons are small targets while labels are large targets.



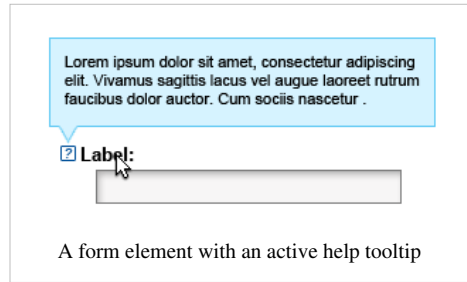
The diagram shows a form element with a small question mark icon to the left of the label "Label:". Below the label is a rectangular input field. The entire element is enclosed in a thin gray border.

A form element with a help trigger

Tooltip Behaviors

Tooltips should always have arrows. The arrows should be obvious and relatively large so that individuals with poor eyesight can readily connect the help text with the element that it is referencing (otherwise, they appear as random blobs of text without context).

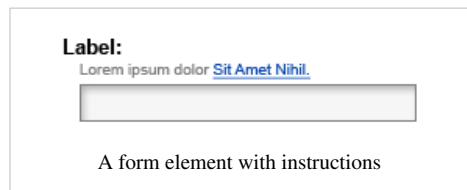
Tooltips should *never* obscure the field that they are attached to. As such, the gravity should be northward. In some cases, a southward gravity may be applied (if the trigger is below the element in question, but this should be rare).



Instructions

Some form elements will benefit by the inclusion of a short set of instructions. Instructions should be placed on a line below the field name and above the input mechanism in a smaller, greyed font.

Field-level instructions should be short and to the point. If additional information is necessary, a link should be provided for more information. This link can activate a tooltip or open further information in a new window or tab. Active links within instructions should *always* be underlined.



Clicking on a "more information" link should **never** load in the same window, causing the user to lose their changes.

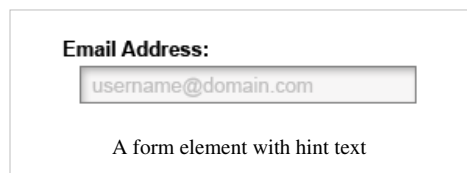
Hint Behaviors

Some form elements can benefit from the presence of data "hints". Hints help the user to decide at a glance the type of data a field takes.

Hint text should be placed within the input mechanism. It should be a light color and the text should be as generic as possible (e.g., "username@domain.com" or "(123) 123-1234").

When a field with hint text achieves *focus* (when the cursor is placed within it), the hint text should become even lighter. Once the user starts typing or inputs data, the hint text should disappear.

If the user clears the field, the hint text should return.



Hint behaviors should be done exclusively with HTML5 placeholder attributes ^[6]. Older browsers will not support this.

Error Handling

See Style guide/Error handling and validation#Error_Notification_Techniques.

Special Function Elements

See Style guide/Specialized form elements.

Client-Side Validation

See Style guide/Error handling and validation#Client-Side_Validation.

Assets

PNG Format



SVG Format



Gallery

Discard Changes

✓ Spell Check

✓ Preview

Save Changes ▶

Save Changes ▶

Save Changes ▶

Label

Option 1

Option 2

Option 3

Option 4

☐ I accept the [terms of service](#).

Label:

Label

Label

☒ Label:

Email Address:

username@domain.com

Email Address:

username@domain.com

Label:

Label

Label:

Label: (Optional)

Label: (Required)

Label:

Option 1

Option 2

Option 3

Option 4

Label:

Label

Allow or Deny:

Denied

Option 1

Create Account

Username

Password

Remember me (up to 30 days)

Forgot your password?

Sign in using your account

Don't have a user account?

Create an Account

Log in

Create Account

Username

Password

Remember me (up to 30 days)

Forgot your password?

Sign in using your account

Don't have a user account?

Create an Account

Log in

References

- [1] <http://uxmovement.com/forms/faster-with-top-aligned-labels/>
- [2] <http://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php>
- [3] <http://www.lukew.com/ff/entry.asp?504>
- [4] <http://css-tricks.com/834-label-placement-on-forms/>
- [5] <http://ux.stackexchange.com/questions/8480/input-form-label-alignment-top-or-left>
- [6] <http://diveintohtml5.org/forms.html#placeholder>

Error handling and validation

This document is a work in progress.

This document is part of the MediaWiki style guide. This section concerns itself with form error handling and validation techniques.

General Notes

When validating forms, there are two possible results for any one form field:

- **Valid** - the data entered by the user is valid and can be saved;
- **Invalid** - the data entered by the user is unacceptable and cannot be saved.

Note that "invalid" data does not necessarily indicate an *error condition* - though most often this is the case. There are myriad ways in which data can be invalid and yet still acceptable for saving.

For example, MediaWiki does not (as of this writing) enforce a valid regular expression for email addresses. A user entering "qasdfasdfas" as an email address may be entering *invalid* data but the system will not throw an error condition.

Server Side vs. Client Side Validation

Field data can be validated on the client side through Javascript. This is often a desired part of a good user experience. However, field data should *always* be validated by the server upon submission, regardless of what client-side validation says about the matter (it is sometimes possible to trick the browser into submitting invalid forms).

As a result of this, one should *never* rely purely on client-side validation for success. All data must be checked by the server on a round trip submission. The server's results (success or failure) should be considered canonical.

Client-Side Validation

Certain field elements can benefit from "client-side validation". In these instances, the value entered by the user is immediately checked for validity through Javascript (either using page-local tests or server requests). These types of fields allow the user to correct known errors without submitting the form and are a powerful usability enhancement.

Live validation events are triggered when the field element loses focus. Validation successes and errors appear in-line with the form element and have icons to draw the user's attention to the result.

Live validation works best with horizontal style message placements.

Client-Side Validation Examples

Here are two examples of live validation behavior.

Password Validity

A common aspect to creating a user account involves creating a password and ensuring its accuracy. This takes two fields: *password* and *confirm password*. Both may be checked for validity.

The "Password" field can be checked for password strength and updated instantly. The "Re-Enter Password" field can be checked to ensure its value is the same as the one entered previously.

Password:

Strength:

Clay

Re-Enter Password:

Matches

The passwords you entered match.

An example of a live update password verification system.

User Name Selection

User name selection can be a difficult process, especially if the desired username is popular.

If the user name is valid and acceptable, then this is validated as positive. If it is invalid (too short, or has special characters), this can be marked invalid with an error message. If it is taken, this can also be displayed (along with optional suggestions, e.g., "Jimbo Wales29").

User Name:
Jorn123456

Available

This user name is available!

User Name:
Foo

Invalid

User names must be greater than 4 characters.

User Name:
Jimbo Wales

Unavailable

This user name is taken. Please select another.

An example of a live update user name selection system. Shows three results.

Error Notification Techniques

Banner Notification

A banner notification that informs the user that an error has occurred should *always* appear, regardless of other error display mechanisms. This banner should inform the user that the action failed and that they (the user) should correct the specific errors below.

Optionally, a list of specific errors may be included in the banner. If the list of errors is greater than 2 or 3, however, the list should be omitted.

If the error is not tied to a specific field (e.g., "Failed to connect to the database"), then the error must be included in the error notification banner. If the errors are only system-level, then the user should not be prompted to fix them (but may be prompted to try again later or contact an administrator).

!

There has been a problem.
One or more errors have occurred. Please correct them below.

An error notification banner

!

There has been a problem.
One or more errors have occurred. Please correct them below.
• Email Address cannot be empty.
• The phone number entered is invalid.

An error notification banner with a list of errors

!

There has been a problem.
A system error has occurred. Please try again later.
• Could not connect to the database (Code #569000)

An error notification banner for a system-level error

Field Local Error Notification

If at all possible, errors should be associated with the fields they are connected to with an error string that helps the user understand what happened and how to correct the error.

Fields with errors should be called out with an obvious icon to the left of the input mechanism. It should be vertically aligned with the input mechanism (below the label and any help trigger icon).

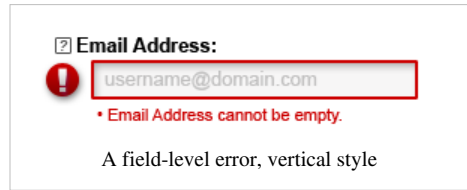
The icon should be at *least* 16x16 pixels in size (a red asterisk is *not* acceptable).

Where possible, the input mechanism should have a red border. This should be at least 2 pixels in thickness but no more than three.

Depending on the form layout and the developer's whim, field level errors can be displayed *horizontally* or *vertically* - or even a mix within the same form.

In *horizontal* alignment, the error list is displayed to the right of the input element. Horizontal alignments are especially good for select boxes or checkbox groupings.

In *vertical* alignment, the error list is displayed directly beneath the element's input mechanism. This is often the most developer-friendly way to display the error.

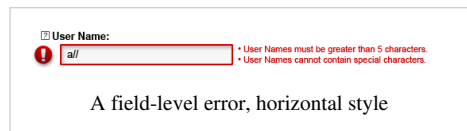


Email Address:

username@domain.com

• Email Address cannot be empty.

A field-level error, vertical style



User Name:

al

• User Names must be greater than 5 characters.
• User Names cannot contain special characters.

A field-level error, horizontal style

Assets

PNG Format



SVG Format



Gallery

Specialized form elements

This document is a work in progress.

This document is part of the MediaWiki style guide. This section concerns itself with specialized interface control elements.

General Notes

From time to time, an interface will call for a specialized control element. These elements duplicate the functionality of existing elements (such as radio button sets) but are easier for the user to understand.

Elements

Toggle Controls

A *toggle* control is a specialized control that can take the place of a single checkbox. Normally, the psychology of a checkbox indicates only a "positive" toggle (the options are "active" or null). In cases where the choice is binary, a toggle is useful.

(Note that a toggle is semantically the same as a radio set with only two options.)



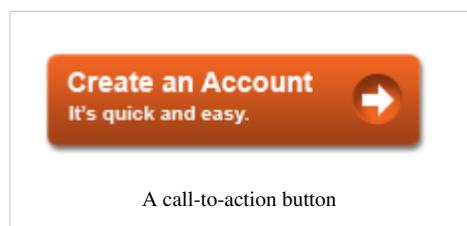
Toggles should have obvious, written values ("enable" and "disable", "on" and "off", etc.). The values should exist within the "bed" of the toggle so as to reduce user confusion as to what the value is set to.

Choice of color for the toggle bed should be carefully considered. Generally, the "negative" state can be left neutral (grey) while the "positive" state should be colored green. In cases where the negative state requires strong consideration (for example, disabling a security measure), the negative bed can should be given an alert color (red or orange).

Call-to-Action Control

A *call-to-action* control is not really a form "element" in that it is part of a form; rather, it is a control that drives the user to engage in an activity (usually filling out another form).

User activities which are considered desirable can be elevated as "calls to action." These controls are interesting to the user in some way or another: they are usually brightly colored (alert colors) and invite exploration and attention.



In most MediaWiki installs, "redlinks" are considered a lightweight form of a call-to-action (note comments below about color). Another canonical example in many web applications is "create an account."

Careful consideration must be applied when choosing color for calls-to-action. In general, "cool" colors (such as blues, greens, and purples) fall "into the background" and are ignored by the user (unless the site is primarily designed with warm colors).

Alert colors (warm) are best. However, color selection is radically limited to the color orange.

- *Yellow* colors as generated by computer monitors are weak choices. Monitors (and televisions) do not easily produce color in the yellow spectrum (pixels are RGB only). Yellow is further a poor choice because of its low

intensity.

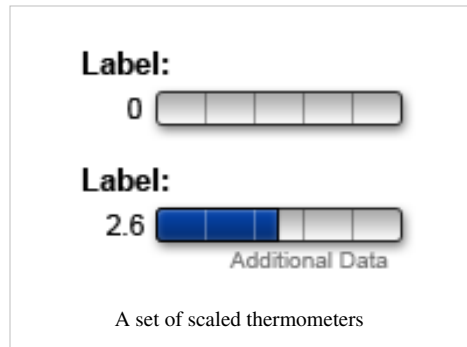
- *Red* colors are psychological stressors to human beings. Humans are hard-wired to recognize the color as "bad".

Scaled Thermometer

A *scaled thermometer* is a display element that is usually used when displaying the average of several values. Scaled thermometers have set values on a pointilist range (e.g., 1 through 5, or 1 through 10, etc.).

When displaying values such in a thermometer, the actual, real value being displayed should be included as a number before the thermometer itself.

If there is additional metadata that could be important (such as the number of entries used in the calculation of the thermometer's value), this should appear below the thermometer, right aligned, and in a smaller font of less-intense color.

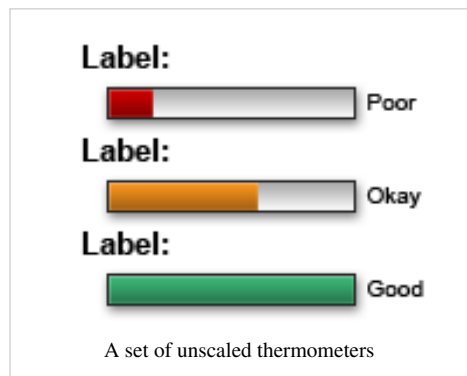


Unscaled Thermometer

Unscaled thermometers are useful when the values to be displayed are vague.

If there is a value judgement that can be applied to the value displayed on the thermometer, both color and text should be used to indicate the value. "Bad" values are red, "moderate" values orange, and "good" values should be green. The text displayed is contextual based upon the display element's purpose.

If there is no value judgement to an unscaled thermometer (there is no "good" or "bad" value), then the color should be a more intense chrome color.

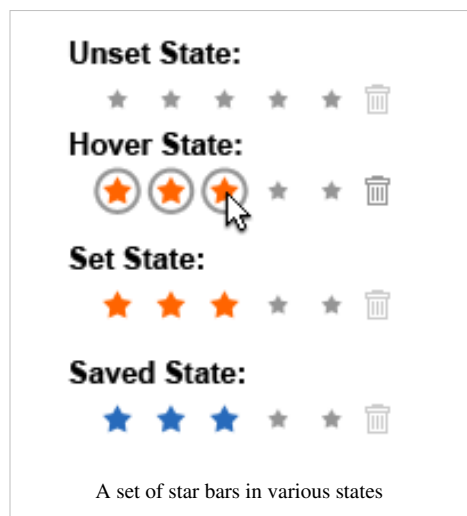


Star Bar

A *star bar* is a control designed for user input within a set "Likert" scale (e.g., numbers between 1 and 5, or 1 and 10, etc.). The user selects the value by clicking on the value they most prefer. A star bar is a modified version of a radiobutton set.

When building a star bar, it is important to understand the various states that a star bar can exist in:

- **Unset State** - this is the star bar before any user interaction
- **Hover State** - this is the star bar's appearance as the user is interacting with it directly
- **Set State** - this is the star bar's appearance once the user has interacted with it but *before* the values have been saved
- **Saved State** - this is the star bar's appearance after results have been saved to the server.



Since color alone should never be an indicator of state, unselected stars should be of smaller size than selected stars.

Assets and gallery

See Also: *Commons:Category:MediaWiki style guide*

Assets

PNG Format



SVG Format



Gallery

Discard Changes

✓ Spell Check

🔍 Preview

Save Changes ▶

Save Changes ▶

→

Save Changes ▶

Label

1

2

3

4

5

☐ I accept the [terms of service](#).

There has been a problem.

One or more errors have occurred. Please correct them below.

✖ Email Address cannot be empty.

There has been a problem.

A system error has occurred. Please try again later.

✖ Email Address cannot be empty.

There has been a problem.

One or more errors have occurred. Please correct them below.

✖ User Name: One or more errors have occurred. Please correct them below.

Email Address:

✖

username@domain.com

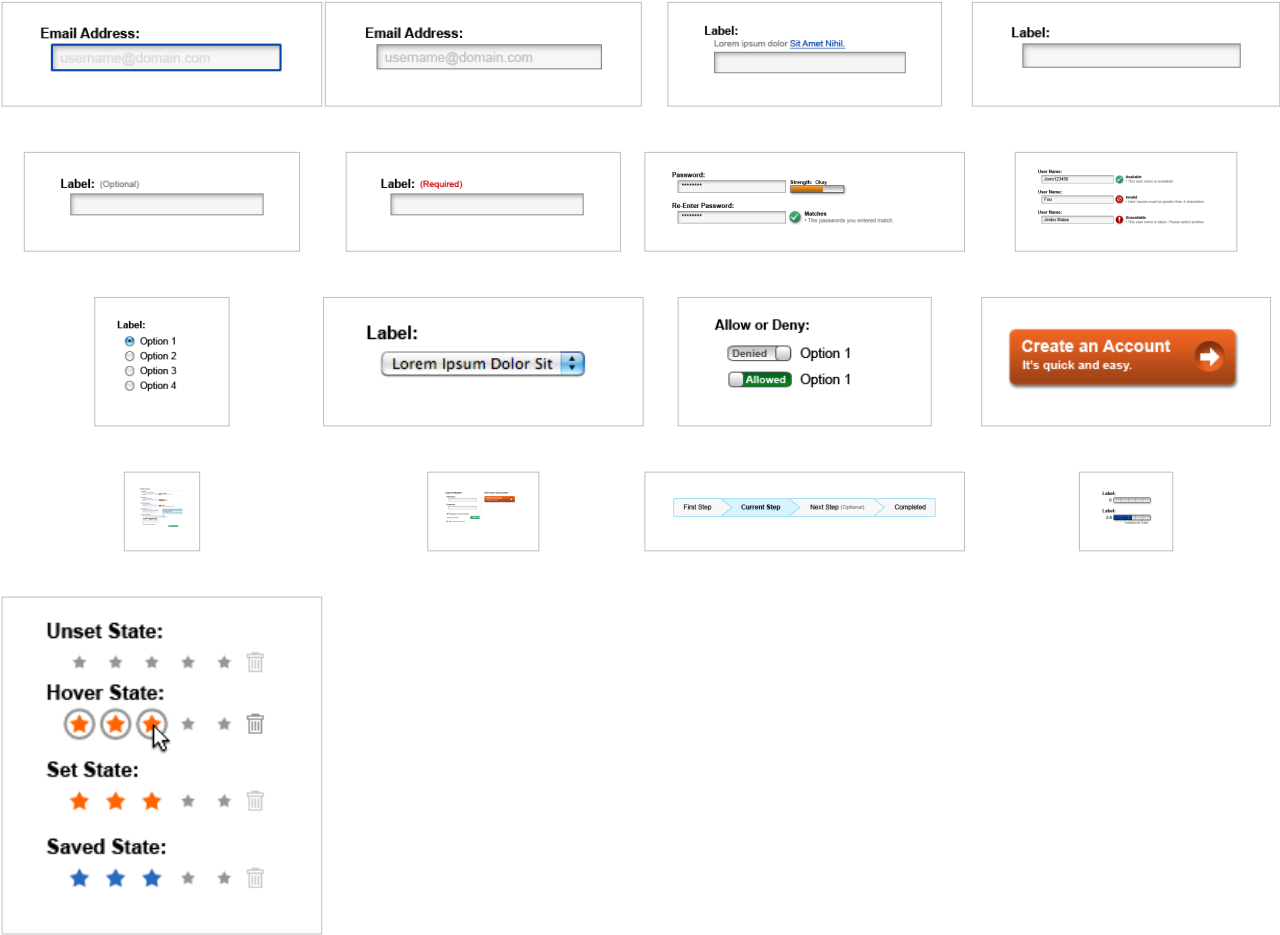
✖ Email Address cannot be empty.

Label:

Label:

Label

Label:



Localisation - Internationalisation (i18n)

Translatewiki.net

Translatewiki.net (formerly known as Betawiki) is a localisation platform for translation communities, language communities, and open source projects. It started out with localisation for MediaWiki. Later support for MediaWiki extensions, FreeCol and other open source projects was added.

Over 300 localisations of MediaWiki and its extensions use translatewiki.net as a convenient translation medium, as well as for coordination and maintenance. Mantis, FreeCol and Mwlib.rl can also be translated in translatewiki.net. More projects are welcome.

You can participate in the translation effort into a language you know. The translations of the MediaWiki interface into new languages for requested projects of the Wikimedia Foundation are also done there.

This project is not part of the Wikimedia Foundation projects, or any foundation or open source project. It is run by Nike and Siebrand, who both are MediaWiki developers and have considerable experience in i18n and L10n. The translation functionality is provided by a MediaWiki extension.

- Link format: `[[translatewiki:]]` or `[[betawiki:]]`

Link

- Translatewiki.net ^[1]

Language:	English
-----------	---------

References

- [1] <http://translatewiki.net>
-

Localisation

This page gives a technical description of MediaWiki's **internationalisation and localisation** (I18N) system, and gives hints that coders should be aware of.

Translation resources

Translatewiki.net

Translatewiki.net supports in-wiki translation of the complete interface. If you would like to have nothing to do with all the technicalities of editing files, subversion, creating patches, this is the place for you.

Finding messages

Manual:System message explains how to find a particular string you want to translate; in particular, note the new qqx trick.

Subversion and Bugzilla

Only few languages are maintained by translators (some Chinese languages), who commit directly to MediaWiki SVN repository. All new efforts should go through previously introduced translatewiki.net

I18n mailing list

You can subscribe to the i18n list; at the moment it is very low-traffic.

Code structure

First, you have a Language object in Language.php. This object contains all the localisable message strings, as well as other important language-specific settings and custom behavior (uppercasing, lowercasing, printing dates, formatting numbers, etc.)

The object is constructed from two sources: subclassed versions of itself (classes) and Message files (messages).

There's also the MessageCache class, which handles input of text via the MediaWiki namespace. And there's the wfMsg*() functions in GlobalFunctions.php. We have large amounts of message retrieval code in GlobalFunctions.php.

General use (for developers)

Language objects

There are two ways to get a language object. You can use the globals \$wgLang and \$wgContLang for user interface and content language respectively. For an arbitrary language you can construct an object by using `Language::factory('en')`, by replacing en with the code of the language. The list of codes is in `languages/Names.php`.

Language objects are needed for doing language specific functions, most often to do number, time and date formatting, but also to construct lists and other things. There are multiple layers of caching and merging with fallback languages, but the details are irrelevant in normal use.

Using messages

MediaWiki uses a *central* repository of messages which are referenced by keys in the code. This is different from, for example, Gettext, which just extracts the translatable strings from the source files. The key-based system makes some things easier, like refining the original texts and tracking changes to messages. The drawback is of course that the list of used messages and the list of source texts for those keys can get out of sync. In practise this isn't a big problem, sometimes extra messages which are not used anymore still stay up for translation.

The message system in MediaWiki is quite complex, a bit too complex. One of the reasons for this is that MediaWiki is a web application. Messages can go through all kinds of processing. The four major ones covering almost all cases are:

1. as-is, no processing at all
2. light wiki-parsing, parserfunction references starting with `{{` are replaced with their results
3. full wiki-parsing

Case 1. is for processing, not really for user visible messages. Light wiki-parsing should always be combined with html-escaping.

Recommended ways

Longer messages that are not used hundreds of times on a page:

- `OutputPage::addWikiMsg`
- `OutputPage::wrapWikiMsg`
- `wfMessage()`

`OutputPage` methods parse messages and add them directly to the output buffer. `wfMessage` can be used when a message should not be added to the output buffer. `->parse()` removes enclosing html tags from the parsed result, usually `<p>...</p>`, but can generate invalid code for example if there is no root tag in parsed result, for example `<p>...</p><p>...</p>`. Usage examples:

```
$out->addWikiMsg( 'foobar', $user->formatNum( count( $items ) ) );
$out->wrapWikiMsg( '<div class="baz">\n$1\n</div>', array( 'foobar', $user->getName() ) );
$text = wfMessage( 'foobar', $language->date( $ts ) )->parse();
```

Other messages with light wiki-parsing can use `wfMsg` and `wfMessage` with `->text()`. **wfMessage** should always be used if the message has parts that depend on linguistic information, like `{{PLURAL:$1}}`. Do not use `wfMsg`, `wfMsgHtml` for those kind of messages! They seem to work but are broken.

```
$out = Xml::submitButton( wfMsg( 'foobar' ) ); # no linguistic
information
$out = Xml::label( wfMessage( 'foobar', $wgLang->formatNum( $count )
)->text() ); # uses plural on $count
```

Some messages have mixed escaping and parsing. Most commonly when using raw links in messages that should not be escaped. The preferred way is to use `wfMessage` with `->rawParams()` for the affected parameters. Be especially wary of using `wfMsgHtml`, it only escapes the message, **not parameters**. This has caused at least one XSS in MediaWiki.

Short list of functions to avoid:

- `wfMsgHtml` (don't use unless you really want unescaped parameters)
- `wfMsgWikiHtml` (breaks up linguistic functions, as does `wfMsg`)
- `OutputPage::parse` and `parseInline`, `addWikiText` (if you know the message, use `addWikiMsg` or `wrapWikiMsg`)

Remember that almost all Xml:: and Html::-functions escape everything fed into them, so avoid double-escaping and parsed text with those.

Adding new messages

1. Decide a name (key) for the message. Try to follow global or local conventions for naming. For extensions, use a standard prefix, preferably the extension name in lower case, followed by a hyphen ("-"). Try to stick to lower case letters, numbers and dashes in message names; most others are between less practical or not working at all. See also [Manual:Coding conventions#Messages](#).
2. Make sure that you are using suitable handling for the message (parsing, { {-replacement, escaping for HTML, etc.)
3. Add it to `languages/messages/MessageEn.php` (core) or your extensions `i18n` file under `'en'`.
4. Take a pause and consider the wording of the message. Is it as clear as possible? Can it be understood wrong? Ask comments from other developers or from localizers if possible. Follow the [#internationalization](#) hints.
5. Add documentation to `MessagesQqq.php` or your extensions `i18n` file under `'qqq'`. Read more about [#message](#) documentation.
6. If you added a message to core, add the message key also to `maintenance/language/messages.inc` (also add the section if you created a new one). This file will define the order and formatting of messages in all message files.

Removing existing messages

1. Remove it from `MessagesEn.php`. Don't bother with other languages - updates from [translatewiki.net](#) will handle those automatically.
2. Remove it from `maintenance/language/messages.inc`

Step 2 is not needed for extensions, so you only have to remove your English language messages from `ExtensionName.i18n.php`.

Changing existing messages

1. Consider updating the message documentation (see [#Adding new messages](#))
2. Change the message key if old translations are not suitable for the new meaning. This also includes changes in message handling (parsing, escaping). If in doubt, ask in [#mediawiki-i18n](#) or in the [Support](#) page at [translatewiki.net](#).
3. If the extension is supported by [translatewiki:translatewiki.net](#), please only change the English source message and/or key. The internationalisation and localisation team will take care of updating the translations, marking them as outdated, cleaning up the file or renaming keys where possible. This also applies when you're only changing things like HTML tags that you could change in other languages without speaking those languages.

Localizing namespaces and special page aliases

Namespaces and special page names (i.e. *RecentChanges* in *Special:RecentChanges*) are also translatable.

Namespaces

To allow custom namespaces introduced by your extension to be translated, create a `MyExtension.namespaces.php` file that looks like this:

```
<?php
/**
 * Translations of the namespaces introduced by MyExtension.
 */
```

```

* @file
*/

$namespaceNames = array();

// For wikis where the MyExtension extension is not installed.
if( !defined( 'NS_MYEXTENSION' ) ) {
    define( 'NS_MYEXTENSION', 2510 );
}

if( !defined( 'NS_MYEXTENSION_TALK' ) ) {
    define( 'NS_MYEXTENSION_TALK', 2511 );
}

/** English */
$namespaceNames['en'] = array(
    NS_MYEXTENSION => 'MyNamespace',
    NS_MYEXTENSION_TALK => 'MyNamespace_talk',
);

/** Finnish (Suomi) */
$namespaceNames['fi'] = array(
    NS_MYEXTENSION => 'Nimiavaruuteni',
    NS_MYEXTENSION_TALK => 'Keskustelu_nimiavaruudestani',
);

```

Then load the namespace translation file in `MyExtension.php` via `$wgExtensionMessagesFiles['MyExtensionNamespaces'] = dirname(__FILE__) . '/MyExtension.namespaces.php';`

When a user installs `MyExtension` on their Finnish (`fi`) wiki, the custom namespace will be translated into Finnish magically, and the user doesn't need to do a thing!

Special page aliases

Create a new file for the special page aliases in this format:

```

<?php
/**
 * Aliases for the MyExtension extension.
 *
 * @file
 * @ingroup Extensions
 */

$aliases = array();

/** English */
$aliases['en'] = array(
    'MyExtension' => array( 'MyExtension' )

```

```
);

/** Finnish (Suomi) */
$aliases['fi'] = array(
    'MyExtension' => array( 'Lisäosani' )
);
```

Then load it in the extension's setup file like this: `$wgExtensionAliasesFiles['MyExtension'] = dirname(__FILE__) . '/MyExtension.alias.php';`

When your special page code uses either `SpecialPage::getTitleFor('MyExtension')` or `$this->getTitle()` (in the class that provides `Special:MyExtension`), the localized alias will be used, if it's available.

Caching

MediaWiki has lots of caching mechanisms built in, which make the code somewhat more difficult to understand. Since 1.16 there is a new caching system, which caches messages either in .cdb files or in the database. Customised messages are cached in the filesystem and in memcached (or alternative), depending on the configuration.

What can be localized

- Namespaces (both core and extensions')
- Weekdays (and abbrev)
- Months (and abbrev)
- Bookstores
- Skin names
- Math names
- Date preferences
- Date format
- Default date format
- Date preference migration map
- Default user option overrides
- Language names
- Timezones
- Character encoding conversion via iconv
- UpperLowerCase first (needs casemaps for some)
- UpperLowerCase
- Uppercase words
- Uppercase word breaks
- Case folding
- Strip punctuation for MySQL search
- Get first character
- Alternate encoding
- Recoding for edit (and then recode input)
- RTL
- Direction mark character depending on RTL
- Arrow depending on RTL
- Languages where italics cannot be used

- Number formatting (commafy, transform digits, transform separators)
- Truncate (multibyte)
- Grammar conversions for inflected languages
- Plural transformations
- Formatting expiry times
- Segmenting for diffs (Chinese)
- Convert to variants of language
- Language specific user preference options
- Link trails, e.g.: [[foo]]bar
- Language code (RFC 3066)

Neat functionality:

- I18N sprintfDate
- Roman numeral formatting

Message parameters

Some messages take parameters. They are represented by \$1, \$2, \$3, ... in the (static) message texts, and replaced at run time. Typical parameter values are numbers ("Delete 3 versions?"), or user names ("Page last edited by \$1"), page names, links, and so on, or sometimes other messages. They can be of arbitrary complexity.

Switches in messages...

Parameters values at times influence the exact wording, or grammatical variations in messages. Not resorting to ugly constructs like "\$1 (sub)page(s) of his/her userpage", we make switches depending on values known at run time. The (static) message text then supplies each of the possible choices in a list, preceded by the name of the switch, and a reference to the value making a difference. This very much resembles the way, parser functions are called in MediaWiki. Several types of switches are available. **These only work if you do full parsing or { {-transformation for the messages.**

...on numbers via PLURAL

MediaWiki supports plurals, which makes for a nicer-looking product. For example:

```
'undelete_short' => 'Undelete {{PLURAL:$1|one edit|$1 edits}}',
```

Language-specific implementations of PLURAL: are found in pages such as LanguageFr.php^[1] (for French; uses singular for 0, code `fr`) or LanguageCs.php^[2] (for Czech, Polish and some other Slavic languages, code `cs`). You should not expect PLURAL to handle fractional numbers (like 44.6) — see bug:28128.

...on user names via GENDER

```
# languages/messages/MessageEn.php
'blocklog-showlog' => 'This user has been blocked
previously.'

# languages/messages/MessageRu.php
'blocklog-showlog' => '{{GENDER:$1|Этот участник уже
блокировался|Эта участница уже блокировалась}} ранее.'
```

If you refer to an user in a message, pass the user name as parameter to the message and add a mention in the message documentation that gender is supported.

...on use context inside sentences via GRAMMAR

Grammatical transformations for agglutinative languages is also available. For example for Finnish, where it was an absolute necessity to make language files site-independent, i.e. to remove the Wikipedia references. In Finnish, "about Wikipedia" becomes "Tietoja Wikipediasta" and "you can upload it to Wikipedia" becomes "Voit tallentaa tiedoston Wikipediaan". Suffixes are added depending on how the word is used, plus minor modifications to the base. There is a long list of exceptions, but since only a few words needed to be translated, such as the site name, we didn't need to include it.

MediaWiki has grammatical transformation functions for over 20 languages. Some of these are just dictionaries for Wikimedia site names, but others have simple algorithms which will fail for all but the most common cases.

Even before MediaWiki had arbitrary grammatical transformation, it had a nominative/genitive distinction for month names. This distinction is necessary if you wish to substitute month names into sentences.

Filtering special characters in parameters and messages

The other (much simpler) issue with parameter substitution is HTML escaping. Despite being much simpler, MediaWiki does a pretty poor job of it. We have a plethora of poorly-named `wfMsg*()` functions, including the multitasking `wfMsgExt()`, with lots of ways to slip up and let through unescaped user input. There may be work done to clean this up at some stage in the future.

Message sources

Messages are obtained from these sources:

- The MediaWiki namespace. It allows wikis to adopt, or override, all of their messages, when standard messages do not fit or are not desired.
 - `MediaWiki:Message-name` is the default message,
 - `MediaWiki:Message-name/language-code` is the message to be used when a user has selected a language other than the wikis default language.
- From message files.
 - MediaWiki itself, and few extensions, use a file per language, called `MessagesZxx.php`, where `zxx` is the language code for the language.
 - Most extensions use a combined message file holding all messages in all languages, usually named after the extension, and having an `.i18n.php` ending.
 - Very few extensions are using another, individual way.

Internationalization hints

Translators ask to consider some hints so as to make their work easier and more efficient. Even if only adding or editing messages in English, one should be aware of the needs of all languages. Messages are translated to more than 300 languages each, which should be done in the best possible way.

There are two main places, where you can find assistance of experienced and knowledgeable people regarding I18n:

- translatewiki.net, ask on their support page
- the `#mediawiki-i18n` `#mediawiki-i18n` ^[3] irc channel on <http://freenode.net> ^[4].

Please do ask them.

Avoid message reuse

The translators encourage reuse avoidance. Although two concepts can be expressed with the same word in English, this doesn't mean they can be expressed with the same word in every language. "OK" is a good example: in English this is used for a generic button label, but in some languages they prefer to use a button label related to the operation which will be performed by the button. If you are adding multiple identical messages, please add message documentation to describe the differences in their contexts. Don't worry too much about the extra work for translators. Translation memory helps a lot in these while keeping the flexibility to have different translations if needed.

Avoid patchwork messages

Languages have varying word orders, and complex grammatical and syntactic rules. Messages put together from lots of pieces of text, possibly with some indirection, are very hard, if not impossible, to translate. Better make messages complete sentences each, with a full stop at the end. Several sentences can usually much more easily be combined into a text block, if needed.

Messages quoting each other

An exception from the rule may be messages referring to one another: *Enter the original authors name in the field labelled "{int:name}" and click "{int:proceed}" when done.* It is safe when a wiki operator alters the messages "name" or "proceed". Without the int-hack, operators would have to be aware of all related messages needing adjustment, when they alter one.

Be aware of PLURAL use on *all* numbers

When a number has to be inserted into a message text, be aware that, some languages will have to use PLURAL on it even if always larger than 1. The reason is that PLURAL in languages other than English can make very different and complex distinctions, comparable to English 1st, 2nd, 3rd, 4th, ... 11th, 12th, 13th, ... 21st, 22nd, 23rd, ... etc.

Do not try to supply three different messages for cases like 0, 1, more items counted. Rather let one message take them all, and leave it to translators and PLURAL to properly treat possible differences of presenting them in their respective languages.

Always include the number as a parameter if possible. Always add `{{PLURAL:}}` syntax to the source messages if possible, even if it makes no sense in English. The syntax guides translators.

You should not expect PLURAL to handle fractional numbers (like 44.5), so it's probably a good idea to round the number to the nearest integer if PLURAL is necessary in the context (bugzilla:28128).

Pass number of list items as parameters to messages talking about lists

At least one language has to use grammar varying with the number of list items when expressing what is listed in a list visible to readers. Thus, whenever your code computes a list, include `count($list)` as parameter to headlines, lead-ins, footers and other messages about the list, even if the count is not used in English. There is a neutral way to talk about invisible lists, so you can have links to lists on extra pages without having to count items in advance.

Separate times from dates in sentences

Some languages have to insert something between a date and a time which grammatically depends on other words in a sentence. Thus they will not be able to use date/time combined. Others may find the combination convenient, thus it is usually the best choice to supply three parameter values (date/time, date, time) in such cases.

Users have grammatical genders

When a message talks about a user, or relates to a user, or addresses a user directly, the user name should be passed to the message as a parameter. Thus languages having to, or wanting to, use proper gender dependent grammar, can do so. This should be done even when the user name is not intended to appear in the message, such as in "inform the user on his/her talk page", which is better made "inform the user on {{GENDER:\$1|his|her|their}} talk page" in English as well.

Avoid {{SITENAME}} in messages

{{SITENAME}} has several disadvantages. It can be anything (acronym, word, short phrase, etc.) and, depending on language, may need {{GRAMMAR}} on each occurrence. No matter what, very likely in most wiki languages, each message having {{SITENAME}} will need review for each new wiki installed. When there is not a general GRAMMAR program for a language, as almost always, sysops will have to add or amend php code so as to get {{GRAMMAR}} for {{SITENAME}} working. This requires both more skills, and more understanding, than otherwise. It is more convenient to have generic references like "this wiki". This does not keep installations from altering these messages to use {{SITENAME}}, but at least they don't have to, and they can postpone message adaption until the wiki is already running and used.

Avoid references to screen layout and positions

What is rendered where depends on skins. Most often screen layouts of languages written from left to right are mirrored compared to those used for languages written from right to left, but not always, and for some languages and wikis, not entirely. Handheld devices, narrow windows, and so on show blocks underneath each other, that appear side to side on large displays. Since user selected and user written javascript gadgets can, and do, hide parts, or move things around in unpredictable ways, there is no reliable way of knowing the actual screen layout.

It is wrong to tie layout information to languages, since the user language may not be the wiki language, and layout is taken from wiki languages, not user languages, unless wiki operators choose to use their home made layout anyways. Acoustic screen readers, and other auxiliary devices do not even have a concept of layout. So, you cannot refer to layout positions in the majority of cases.

We do not currently have a way to branch on wiki directionality (bug 28997)

The upcoming browser support for East and North Asian top-down writing^[5] will make screen layouts even more unprecitable.

Have message elements before and after input fields

This rule has yet to become de facto standard in MediaWiki development

While English allows efficient use of prompting in the form "item colon space input-field", many other languages don't. Even in English, you often want to use "Distance: ____ feet" rather than "Distance (in feet): ____". Leaving `<textarea>` aside, just think of each and every input field following the "Distance: ____ feet" pattern. So:

- give it two messages, even if the 2nd one is most often empty in English, or
- allow the placement of inputs via \$i parameters.

Avoid untranslated HTML markup in messages

HTML markup not requiring translation, such as enclosing `<div>`s, rulers above or below, and similar, should usually better not be part of messages. They unnecessarily burden translators, increase message file size, and pose the risk to accidentally being altered in the translation process.

Messages are usually longer than you think!

Skimming foreign language message files, you find messages almost never shorter than Chinese ones, rarely shorter than English ones, and most usually much longer than English ones.

Especially in forms, in front of input fields, English messages tend to be terse, and short. That is often not kept in translations. Especially genuinely un-technical third world languages, vernacular, medieval, or ancient languages require multiple words or even complete sentences to explain foreign, or technical, prompts. E.g. "TSV file:" may have to be translated as: *"Please type a name here which denotes a collection of computer data that is comprised of a sequentially organized series of typewritten lines which themselves are organized as a series of informational fields each, where said fields of information are fenced, and the fences between them are single signs of the kind that slips a typewriter carriage forward to the next predefined position each. Here we go: _____ (thank you)"* — admittedly an extreme example, but you got the trait. Imagine this sentence in a column in a form where each word occupies a line of its own, and the input field is vertically centered in the next column. :-(

Avoid using very close, similar, or identical words to denote different things, or concepts

For example, pages may have older *revisions* (of a specific date, time, and edit), comprising past *versions* of said page. The words *revision*, and *version* can be used interchangeably. A problem arises, when versioned pages are revised, and the revision, i.e. the process of revising them, is being mentioned, too. This may not pose a serious problem when the two synonyms of "revision" have different translations. Do not rely on that, however. Better is to avoid the use of "revision" aka "version" altogether, then, so as to avoid it being misinterpreted.

Basic words may have unforeseen connotations, or not exist at all

There are some words that are hard to translate because of their very specific use in MediaWiki. Some may not be translated at all. For example "namespace", and "apartment", translate the same in Kölsch. There is no word "user" relating to "to use something" in several languages. Sticking to Kölsch, they say "corroborator and participant" in one word since any reference to "use" would too strongly imply "abuse" as well. The term "wiki farm" is translated as "stable full of wikis", since a single crop farm would be a contradiction in terms in the language, and not understood, etc.

Expect untranslated words

This rule has not yet become de facto standard in MediaWiki development

It is not uncommon that computerese English is not translated and taken as loanwords, or foreign words. In the latter case, technically correct translations mark them as belonging to another language, usually with appropriate html markup, such as ` ... `. Thus make sure that, your message output handler passes it along unmolested, even if you do not need it in English, or in your language.

Permit explanatory inline markup

This rule has yet to become de facto standard in MediaWiki development

Sometimes there are abbreviations, technical terms, or generally ambiguous words in target languages that may not be immediately understood by newcomers, but are obvious to experienced computer users. So as not to create lengthy explanations causing screen clutter, it may be advisable to have them as annotations shown by browsers when you move the mouse over them, such as in:

```
mḍwwer 90° <abbr title="Ġks (t-tijah) Ġaqarib s-Saġa">ĠĠS</abbr>
```

giving:

```
mḍwwer 90° ĠĠS
```

explaining the abbreviation for "counter clockwise" when needed. Thus make sure, your output handler accepts them, even if the original message does not use them.

Symbols, colons, brackets, etc. are parts of messages

Many symbols are translated, too. Some scripts have other kinds of brackets than the Latin script has. A colon may not be appropriate after a label or input prompt in some languages. Having those symbols included in messages helps to better and less anglo-centric translations, and by the way reduces code clutter.

Do not expect symbols and punctuation to survive translation

Languages written from right to left (as opposed to English) usually swap arrow symbols being presented with "next" and "previous" links, and their placement relative to a message text may, or may not, be inverted as well. Ellipsis may be translated to "etc." or to words. Question marks, exclamation marks, colons do appear at other places than at the end of sentences, or not at all, or twice. As a consequence, always include all of those in your messages, never insert them programmatically.

Use full stops

Do terminate normal sentences with full stops. This is often the only indicator for a translator to know that they are not headlines or list items, which may need to be translated differently.

Link anchors

Wikicode of links

Link anchors can be put into messages in several technical ways:

1. via wikitext: ... `[[a wiki page|anchor]]` ...
2. via wikitext: ... `[some-url anchor]` ...
3. the anchor text is a message in the MediaWiki namespace. *Avoid it!*

The latter is often hard or impossible to handle for translators, avoid patchwork messages here, too. Make sure that "some-url" does not contain spaces.

Use meaningful link anchors

Care for your wording. Link anchors play an important role in search engine assessment of pages, both the linking ones, and the ones linked to. Make sure that, the anchor describes the target page well. Do avoid commonplace and generic words! For example, "Click here" is an absolute nogo, since target pages never are about "click here". Do not put that in sentences around links either, because "here" was not the place to click. Use precise words telling what a user will get to when following the link, such as "You can upload a file if you wish."

Avoid jargon and slang

Avoid developer and power user jargon in messages. Try to use as simple language as possible.

One sentence per line

Try to have one sentence or similar block in one line. This helps to compare the messages in different languages, and may be used as an hint for segmentation and alignment in translation memories.

Be aware of whitespace and line breaks

MediaWiki's localized messages usually get edited within the wiki, either by admins on live wikis or by the translators on translatewiki.net. You should be aware of how whitespace, especially at the beginning or end of your message, will affect editors:

- Newlines at the beginning or end of a message are fragile, and will be frequently removed by accident. Start and end your message with active text; if you need a newline or paragraph break around it, your surrounding code should deal with adding it to the returned text.
- Spaces at the beginning or end of a message are also likely to end up being removed during editing, and should be avoided. If a space is required for output, usually your code should be appending it or else you should be using a non-breaking space such as ` `; (in which case check your escaping settings!)
- Try to use literal newlines rather than `"\n"` characters in the message files; while `\n` works in double-quoted strings, the file will be formatted more consistently if you stay literal.

Use standard capitalization

Capitalization gives hints to translators as to what they are translating, such as single words, list or menu items, phrases, or full sentences. Correct (standard) capitalization may also play a role in search engine assessment of your pages. If you really need to emphasise something with capitals, use CSS styles to do so. For instance, the HTML attributes `style="text-transform:uppercase"` (uppercase) or `style="font-variant:small-caps"` (Small Caps) will do. Since these may be adjusted to something else during translation, most specifically for non-Latin scripts, they need to be part of the messages and must not be added programmatically.

Emphasis

In normal text, emphasis like **boldface** or *italics* and similar should be part of message texts. Local conventions on emphasis often vary, especially some Asian scripts have their own. Translators must be able to adjust emphasis to their target languages and areas.

Keeping messages centralized and in sync

English messages are very rarely out of sync with the code. Experience has shown that it's convenient to have all the English messages in the same place. Revising the English text can be done without reference to the code, just like translation can. Programmers sometimes make very poor choices for the default text.

Message documentation

There is a pseudo-language, having the code `qqq` (message documentation). It is one of the ISO 639 codes reserved for private use. There we do not keep translations of each message, but collect English sentences *about each message*; telling us where it is used, giving hints about how to translate it, enumerate and describe its parameters, link to related messages, etc.. In translatewiki.net, these hints are shown to translators when they hit the `""` button for messages.

Programmers must play a role in message documentation. Whenever a message is added to the software, a corresponding `qqq` entry must be added as well. Useful information includes:

1. message handling (parsing, escaping, plain text)
2. type of parameters with example values
3. where the message is used (pages, locations in the user interface)
4. how the message is used where it is used (a page title, button text, etc..)
5. what other messages are used together with this message, or which other messages this message refers to
6. anything else which could be understood when the message is seen on the context, but not when the message is displayed alone (which is the case when it is being translated)
7. if the message has special properties, like is a page name, or if it should not be a direct translation
8. parts of the message which must not be translated, such as generic namespace names or URLs
9. You can link to other messages by using `{{msg-mw|message key}}`. Please do this if parts of the messages come from other messages (if it cannot be avoided), or if some messages are shown together or in same context.

License

Any edits made to the language must be licensed under the terms of the GNU General Public License (and GFDL?) to be included in the MediaWiki software.

History

With MediaWiki 1.3.0 a new system has been set up for localizing MediaWiki. Instead of editing the language file and asking developers to apply the change, users can now edit the interface strings directly from their wikis. This is the system in use as of August 2005. People can find the message they want to translate in `Special:AllMessages` and then edit the relevant string in the `MediaWiki:` namespace. Once edited, these changes are live. There is no more need to request an update, and wait for developers to check and update the file.

The system is great for Wikipedia projects; however a side effect is that the MediaWiki language files shipped with the software are no longer quite up-to-date, and it is harder for developers to keep the files on meta in sync with the real language files.

As the default language files do not provide enough translated material, we face two problems:

1. New Wikimedia projects created in a language which have not been updated for a long time, need a total re-translation of the interface.
2. Other users of MediaWiki [not Wikimedia projects] are left with untranslated interfaces. This is especially unfortunate for the smaller languages which don't have many translators.

This is not such a big issue anymore, because translatewiki.net is advertised prominently and used by almost all translations. Local translations still do happen sometimes.

Missing

This section is missing about the changes in the i18n system related to extensions. The format was standardized and messages are automatically loaded. See *Message sources* above.

References

- [1] <http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3/languages/classes/LanguageFr.php>
- [2] <http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3/languages/classes/LanguageCs.php>
- [3] <irc://irc.freenode.net/>
- [4] <http://freenode.net/>
- [5] <http://dev.w3.org/csswg/css3-writing-modes/>

Newcomers

How to become a MediaWiki hacker

This article is written to help novice developers learn the skills needed to contribute to MediaWiki development.

If you are an experienced developer, visit the **developer hub** instead.

Overview

The MediaWiki software is written in PHP and uses the MySQL database. Both have been ported to a variety of operating systems, including, but not limited to, most Unix variants (Linux, Mac OS X, etc.) and Microsoft Windows. It is possible to install and use MediaWiki on Linux, Mac OS X and Windows. Note: if you do use Windows, certain features involving external utilities will be unavailable, or only available with special downloads and configuration. You'll sometimes see operating system-dependent bugs, so it is best to have some knowledge of the difference between the various platforms regardless of which operating system you develop on.

Here's a short overview of MediaWiki's architecture.

Getting help

The fastest way to get help as you get started is our Internet Relay Chat channel, [#mediawiki](#) ^{connect [1]}.

The PHP programming language

If you have no knowledge of PHP (PHP stands for "PHP: Hypertext Preprocessor") but know how to program in other object-oriented programming languages, have no fear, PHP will be easy for you to learn.

If you have no knowledge of PHP or other object-oriented programming languages, you should familiarize yourself with concepts such as classes, objects, methods, events and inheritance.

If you have no knowledge of any programming language, PHP is a good language to start with, as it is reasonably similar to other modern languages, although it is specific in the way it is executed.

PHP scripts can run from the command line, or a window manager is enough to call the interpreter. e.g., (on Linux/UNIX):

```
/usr/bin/php -q < phpshell.php
```

The script `maintenance/eval.php` in MediaWiki provides a basic PHP interpreter with MediaWiki objects and classes loaded.

Usually, for websites, a PHP script is executed when you request a file with the ".php" extension (among others) from a webserver. As you do that, the web server, in our case Apache, calls the PHP interpreter (which may be built into the webserver), interprets the PHP file and returns the result to your browser. The PHP file can contain both regular HTML and PHP code, which makes it relatively simple to add dynamic functionality to a static webpage.

Related links

- PHP tutorial ^[2] (available in many different languages)
- The PHP manual ^[3] (available in many different languages)
- PHP at Wikibooks

Database

MediaWiki currently uses MySQL as the primary database backend. It also supports other DBMSes, such as PostgreSQL and SQLite. However, almost all developers use MySQL and don't test other DBs, which consequently break on a regular basis. You're therefore advised to use MySQL when testing patches, unless you're specifically trying to improve support for another DB. In the latter case, make sure you're careful not to break MySQL (or write queries that are horribly inefficient in it), since that's what everybody else uses.

Although the Wikimedia Foundation has now moved on from MySQL 4.0, it's important to not intentionally break MySQL 4.0 support. MySQL 4.0 is missing a lot of features of later MySQL versions (never mind other DBMSes): if you aren't sure, double-check in the manual ^[4] first! The most commonly used feature missing from MySQL 4.0 is subqueries; don't use those outside of code specific to a non-MYSQL DBMS.

Installing MediaWiki

Get the latest sources from Subversion, our source control system, before creating patches. See Download from SVN for how to get the sources from SVN.

Follow the instructions in the INSTALL file in the source. You could also read the installation guide.

It's not necessary to download Wikipedia database dumps in order to develop MediaWiki features. In fact, in many cases it's easier to use a near-empty database with a few specially-crafted test pages. However, if for some reason you want to have a copy of Wikipedia, you can get a dump from meta:data dumps.

You may also find that you get an error complaining that access was denied to the wiki database. Make sure that you have created a file `AdminSettings.php` in your top-level MediaWiki install directory (the same place as `LocalSettings.php` is found). An `AdminSettings.sample` file is provided for you to customise - make sure your MySQL administration username and password is set correctly. See Manual:Upgrading for more details.

Rebuilding the link tables may take a long time, particularly if you've installed the English Wikipedia database, which is quite big. (Note also that you can skip the old table if you wish.) See Manual:Database layout on what `rebuildall.php` is good for.

Note that if you want to create a *public mirror* of Wikipedia, this probably isn't the best way to go about it. If you do set up a mirror this way, *please* tweak the code to note that you're looking at a mirror and include links back to the main site. See Forks and Mirrors for more info.

The MediaWiki codebase

The MediaWiki codebase is large and ugly. Don't be overwhelmed by it. When you're first starting off, aim to write features or fix bugs which are constrained to a small region of code.

Browse through the list of important files at [Manual:Code](#). For more detailed information, browse the generated documentation ^[5] (warning: huge page will be loaded).

Your first feature

Here are some ideas:

- Code something that interests you;
- Fix an annoying little bug that nobody else could be bothered with;
- Write a special page to provide some handy information;
- Write a parser hook;
- Write a simple extension.

For more specific suggestions, please come and talk to the developers on [#mediawiki](#) ^{connect} ^[1]. If you already have an idea for a feature you want to implement, it's also a good idea to talk to a senior developer before you start, especially if you're not sure how your feature will affect other parts of the code.

Make your changes against the trunk in Subversion, not a branch, and try to follow the coding standards (if you're writing a brand-new file, [styleize.php](#) ^[1] can help).

When you have a feature ready to go, post a patch in Bugzilla, mark it with the "patch" and "need-review" keywords, and ping the Bugmeister to have it reviewed and committed. This can be a slower process than just committing it yourself, but by doing it once or twice you demonstrate your good faith, and your ability to write reasonably stable code. Relatedly, before you commit your feature, make sure it can be disabled easily.

When you are ready to write a new MediaWiki extension and you'd like to get it deployed on Wikimedia project servers, read [Writing an extension for deployment](#).

Testing

Use `E_STRICT` in your `php.ini` to have unnecessary warnings and notices reported early.

When adding features, it's vital to verify you didn't break existing functionality. The usual tool for this is automated testing frameworks. MediaWiki's test suite is still relatively sparse. We have three kinds of tests:

- Parser tests (see `tests/parserTests.php` ^[6]), which only test the parser. Try running `php tests/parserTests.php --quick --quiet` to see how those work. Everything should pass, in theory. You can add new tests or fix existing ones by editing `tests/parserTests.txt`.
- PHPUnit-based unit tests in the `tests/phpunit` ^[2] directory. They are typically run through the `phpunit.php` script invoked from the aforementioned directory. These tests also include ordinary parser tests, though `parserTests.php` probably works faster. See [Manual:PHP unit testing for PHPUnit setup](#) instructions and further details.
- Selenium tests are in directory `tests/selenium` ^[7].

Anyway, if you can't write an automatic test, do manual testing. If you cause breakage too often, people will get annoyed at you, especially if it isn't caught until it goes live on Wikipedia. Revocation of commit access has been threatened in the past occasionally. At the very least, expect serious indignation if you check in syntax errors – try at least loading your wiki, or

```
php maintenance/checkSyntax.php --modified.
```

Turning `display_startup_errors` on

`display_startup_errors` is off by default on the toolserver. Turning it on within the program under test is too late! So create the following stub and execute that instead.

```
error_reporting(0x7FFF ^ (E_NOTICE | E_STRICT));
ini_set('display_startup_errors',1);
ini_set("display_errors",1);
include("program_currently_under_test.php");
```

Posting a patch

If you have created and tested a patch (for example to fix an annoying little bug), get a unified diff^[8] of the modified file by using:

```
svn diff path/to/modified_file.php > my.patch
```

Then post the patch as an attachment to the appropriate bug report in Bugzilla, and add the `patch` and `need-review` keywords to it so developers know to review it and respond to you.

References

- [1] <http://webchat.freenode.net/?channels=#mediawiki>
- [2] <http://www.php.net/manual/en/tutorial.php>
- [3] <http://www.php.net/manual/en/>
- [4] <http://dev.mysql.com/doc/refman/4.1/en/>
- [5] <http://svn.wikimedia.org/doc/>
- [6] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/parserTests.php?view=markup>
- [7] <http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/tests/selenium>
- [8] https://en.wikipedia.org/wiki/Diff#Unified_format

Commit access

This article is a guide to applying for and using MediaWiki commit access.

Requesting commit access

Prerequisites

Before applying for commit access, you should typically have:

- A demonstration that your request is made in good faith. For example, someone may be employing you to work on MediaWiki, or you may be known to the Wikimedia volunteer community by way of past work.
- Plans to contribute to MediaWiki or extensions in a substantial way.
- Programming skills appropriate for the type of work you propose to do. Our community will help new participants, especially with MediaWiki-specific skills, but we don't have time to train programmers from scratch.
- An account on this wiki (www.mediawiki.org) with an authenticated e-mail address set in your Special:Preferences.

Request format

Commit access requests should be sent to commit-access-requests@wikimedia.org ^[1].

Your request should contain the following information:

Requested user name

Your user name must match the following regular expression `/^[0-9a-z_-\.\]{1,17}$/`. That is, it may only contain from 1 to 17 numbers and lower case ASCII letters, as well as the `.` (full stop), `-` (hyphen-minus) and `_` (low line) characters.^[2] Additionally, the user name should not conflict with the existing user names at <http://svn.wikimedia.org/users.php>.

Your www.mediawiki.org user name

You must be registered and have e-mail features enabled to receive code review feedback.

What you expect to work on

Which extensions, core modules, etc.

A link to your public key (public key openssh format)

Your SSH key will be used to authenticate you when you commit changes.

For security reasons, you should temporarily host your public key on your personal webserver and supply the *link* to it. Less preferably, post it to a wiki by a user name known to us. The key should be in the **openssh** format (*not* SSH2 format ^[3]), see example :

```
ssh-rsa AAAB3NzaC1yc2EAAAAB3FWqgypbL ... Lgfe180W3Bm9sbb+5JcsHV comment...
```

in plain text, and not embedded in HTML. `ssh-keygen` (for Unix-like operating systems) and `PuTTYgen` ^[4] (for Windows-family operating systems)^[5] ^[6] are examples of programs that can generate a key for you.



Warning: Do *not* send a key as an e-mail attachment, as unsigned e-mail is not a secure means of delivering your SSH key. Dropbox, Megaupload, Rapidshare, and the like are also unacceptable.

Past work on MediaWiki

Patches, extensions, bug reports, etc. Please provide links or attachments to specific patches, changesets, or codebases **that you wrote**; links are preferred.

Past programming experience

Other open source projects, etc.

Note: It may take up to three weeks for your request to be processed.

Getting started and set up

Once you have an account, you will need to follow some steps before you can commit anything.

1. Read Subversion to get an overview of the Subversion source control system.
2. Read Subversion/auto-propos and do exactly what it says to set up auto-propos.
3. Create a USERINFO file.
4. Checkout using `svn+ssh://` instead of `http://` , or it simply won't work (you may get a 403 Forbidden unexpected return value).

Note: If you are prompted for a password during checkout and your SSH key doesn't need a passphrase, this may be an indication that your account has not been set up correctly. *If you're using Windows, check that Pageant is running with your private key stored.*

5. Check out two copies of MediaWiki: one instance for development and testing, and another instance that you will apply patches to and commit from. This structure makes it easier to commit clean patches. To do this, use:

```
svn co svn+ssh://username@svn.wikimedia.org/svnroot/mediawiki/trunk/phase3
wikimedia-dev

svn co svn+ssh://username@svn.wikimedia.org/svnroot/mediawiki/trunk/phase3
wikimedia-commit
```

6. Have a coder link your SVN account to your wiki account, so that you receive emails when somebody comments on your commits. If you're feeling bold, you could instead ask a coder to make you a coder; you can then link your account yourself at `Special:Code/MediaWiki/author/<your name>`.

Updating/replacing your SSH key

- Go to this subpage of your personal user page on this wiki.
- Click "create". Paste in the text of your new key, then click "save page".
- You will be redirected back to the page, and it will now have your key on it. Copy the URL to your clipboard.
- Send an email to `commit-access-requests@wikimedia.org`^[1] of the form:

Hi,

I'm a MediaWiki developer. Could you please update my ssh key for the user name *insert username here*? My SSH key is at *paste URL here*.

Thanks,

your name

Guidelines for applying patches

- Remember that **trunk is supposed to be *live code*** ^[7] -- keeping things up to date and ready to run at all times is our biggest method of quality control. Code *must work* at all times, except of course when it's broken by accident.
- Always **update the RELEASE-NOTES** ^[22].
- Consider possible **security implications** (e.g. see this security note for extension authors ^[8]).
- **Do not backport** any new features to the stable trunk without prior permission ^[9]. In fact, best to just leave the stable trunk alone. Leave all backporting of fixes to the release manager ^[10], and save yourself grief. Things should only be backported if they are a security fix, or are vital for the software to run ^[11]. Also, new features *don't* go into maintenance releases of old versions ^[12].
- **Don't touch the release tags at all** ^[13]. Ever ^[14]!
- **Try not to break things**, or people may get cranky. For example, if you are changing the parser, you can run `parserTests` before and after to help determine if your change introduced any new breakages, like so:

```
cd tests
php parserTests.php --quiet --quick --color=no
```

... and compare the failures before with the failures after, to check that there were no regressions.

- **Test all patches thoroughly** before applying them to SVN HEAD.
- Try not to make any big changes *right before the tree is branched into a stable release* (this currently happens about once every 3 months).
- **Try to avoid introducing new preferences**, if possible, by supplying sane defaults ^[15]. Remember that new preferences should very rarely be added ^[16].
- Try to follow MediaWiki's Coding conventions where possible. Read them every so often because they might be updated.
- MediaWiki caches parser output and revision diffs ^[17]. Take into account that Wikimedia and other sites use HTTP caching (with Squid or Varnish) in front of the webserver. Try not to introduce code that makes caching difficult.
- I18n (internationalisation) and new message texts must be added (or changed) on translatewiki.net, not in the SVN. The changes will be committed to Wikimedia SVN by the translatewiki.net developers in due time.
- Before committing someone else's software to Wikimedia SVN, it is best to discuss with them first. Ideally, invite the original developer to apply for commit access and wait for that request to be fulfilled so they can commit it themselves.

To commit to SVN

Once you have done all the above steps, you are ready to make your first commit. You should join MediaWiki developers' IRC channel ([#mediawiki](http://irc.freenode.net)) ^[18]. This way you people can give you direct feedback about your commits and get to know you and you them.

Always make sure to have your local copy as close as possible to the head version (update), check the files you have changed, added, deleted or not committed (status), and always compare your local copy including your changes with the latest version in the repository (difference local to HEAD). Commit only, if everything is as expected:

```
cd my_workingcopyfolder
svn up
svn status
svn diff -rHEAD > my_workingcopyfolder-to-head.diff
svn commit -m "my message"
```

To prepare a commit, run `svn diff` List of files you want to commit. If you don't give filenames, `svn` picks up all files recursively from your current directory. Review the diff once more for any unrelated changes and bugs in the code. When you are ready run `svn ci` Same list of files. Your default text editor will pop up and ask for a commit summary. The following tips help to make good commit summaries:

- Say what you did and why it's a good thing or needed.
- The first line should contain concise summary of the commit.
- In the following lines you can go into details of what and why. For bug fixes, try to give clear instructions to reproduce the bug, so reviewers can confirm that the commit does fix the problem.
- Reference related bugs with `bug #####` and revisions with `r#####` for each bug and revision. You need to use that exact syntax so that those will become links *automatically*.
- Once you have written the whole summary, read it once to catch any errors.

If you want to write the summary in steps, you can use an ordinary file and the `--file` for the `svn ci` command.

Then you should get output back like this:

```
Sending          RELEASE-NOTES
Sending          includes/User.php
Transmitting file data ..
Committed revision 16794.
```

Then you can close any affected bugs (if applicable) in bugzilla ^[19], using a line like "Fixed in r#####."

Then hang around on #mediawiki for at least a few minutes, so that people can find you in case you broke anything. You might also get feedback on Code review, so be sure that your user account is linked and that you allow email notifications.

Some useful resources or commands

- To see what changed in a particular revision number, go to <http://svn.wikimedia.org/viewvc/mediawiki?view=rev&revision=16789> and replace the revision number in the URL as appropriate.
- The "svn status" command will show you any files that differ between your local version and the central version.
- The "svn diff includes/file_you_changed.php" command will show what exactly has been changed in a certain file, relative to the last checked out version.
- If you want to see what changed between two revisions from the command line:

```
svn diff --revision 17109:17114
```

- Show differences between your local copy and the current code in the repository

```
svn diff -rHEAD
```

- You should probably subscribe to the wikitech mailing list ^[20].
- If you want to see what's changed recently, you can use the CVS mailing list ^[21].
- Sometimes there is a small bit of lag between commits, and getting notified via CVS email. There is no lag however with SVN, so to see what's changed recently, or even not-so-recently, you can use the "svn log" command. For example this will show all changes between the specified revision numbers, and details of which files were changed:

```
svn log --revision 17109:17114 --verbose
```

- You can always ask questions on the #mediawiki IRC channel. Also if the CIA-7 (or 60, or 57, or whatever the number may be) bot is working, it will print out notifications of SVN commits to the #mediawiki channel as they

happen.

- If you commit a change, and it turns out to be completely broken, then you can revert it by doing something like this (undoing changes explained in detail ^[22]).

Note: You can add the merge option `--dry-run` to preview the changes as it does not apply changes to the working copy.

```
svn up

svn merge -r REVISION_NUM_YOU_WANT_TO_UNDO:REVISION_NUM_TO_REVERT_TO .

svn status

svn diff

svn commit --message="Self-revert accidental breakage" includes/file-you-changed.php includes/another-file-you-changed.php
```

For reverting a single version, you can use the changeset notation (take note of the - sign before the revision number to apply the reverse difference)

```
svn up

svn merge -c -REVISION_YOU_WANT_TO_UNDO .

svn status

svn diff

svn commit --message="Self-revert accidental breakage" includes/file-you-changed.php includes/another-file-you-changed.php
```

- To view a specific bug in bugzilla, go to http://bugzilla.wikimedia.org/show_bug.cgi?id=#### (replace #### with the bug number in the URL as appropriate).
- When closing a bug in bugzilla, include a comment with something like "Fixed in r####" in it - it autolinks the revision, and it helps us to keep track of changes related to bugs.
- When committing a change which fixes a bug, always include the bug number in the commit summary.

Commit Summaries

When committing to the MediaWiki SVN repository, please try and post an informative commit summary.

If you are doing work towards (or to fix a bug), please use the bug #### syntax to link it. If this commit follows on, reverts, or is related to other commits, please reference them using the r#### syntax.

Try and ensure that your first line of your commit summary is as informative as possible. Due to the use of Code review, this only displays the first line of the commit summary in the overview. Try to be concise in this first line, and then elaborate in further detail below.

In the case of controversial, or non obvious (why you are doing them/why this is a benefit) changes, please link to any relevant discussions (if appropriate) - Bug comments, Bugzilla bugs, mailing list posts. Else, if these are not the case, explain why you are doing this commit (an essay isn't required, but "Fix bug" isn't enough information).

Magic linking

Syntax like

- bug 12345
- r12345
- follow up patch to r12345

in a commit message or code comment automatically creates a link to the specific bug or revision, or set a follow-up indication at the specific revision it follows up. When used in a commit message specifically, this syntax also creates links between revisions (to note follow-up revisions, for example).

Going live

Code in Subversion won't immediately be live on Wikimedia wikis like Wikipedia. This is to avoid immediate breakage of the sites and to allow for code review to take place.

When a system administrator (typically Brion Vibber or Tim Starling in this case) has reviewed code changes, they will perform a standard `svn update` on the production cluster. At this point, changes are live on `http://test.wikipedia.org`, and last-minute checking can be done to ensure nothing has been broken under Wikimedia's idiosyncratic configuration.

If everything appears okay, then changes will be synchronised to application servers. This is also known as "scapping", after the "scap" script which does this.

To find out who last modified a particular line of code

Run a command like this (using the `includes/Parser.php` file as an example) :

```
svn blame includes/Parser.php | less
```

... and then search for the line by typing `/` + your search term (e.g. the function name). It may take about 20 seconds for the `svn blame` output to be generated.

This will show which revision a line was modified in, and by whom, like so:

```
4452      timwi      $argc = count($args);
9860 kateturner
4452      timwi      for ( $i = 0; $i < $argc-1; $i++ ) {
4904      hashar          if ( substr_count ( $args[$i], '[' ) != substr_count ( $args[$i], ']' ) ) {
4904      hashar          $args[$i] .= '|' . $args[$i+1];
```

... you can then look up that particular revision number using the SVN web interface, to see what the commit log message was to get an explanation of why a change was made. If necessary, you can then ask the user; there is a list of SVN committers^[23] in the SVN server with contact information.

Setting up a non-standard port

Tip: Only needed if you use a non-standard port for SSH normally, otherwise ignore this. You can force SVN to use the right port number by adding this to `~/.subversion/config` :

```
[tunnels]
ssh = $SVN_SSH ssh -p 22
```

Branching and merging

You may wish to do ongoing experimental work in a branch, so that the development trunk remains stable until your change is ready to merge. Branches are also used for the quarterly releases, so that additional bug fix releases can be made easily.

See the Subversion/branching guide for more...

Notes

- [1] <mailto:commit-access-requests@wikimedia.org>
- [2] While valid Unix user names can contain many other characters other than `^[0-9a-z_\-\.]{1,17}$`, we should avoid the use of metacharacters that can cause us trouble in some contexts if we forget to escape any meta-characters.
- [3] <http://www.ietf.org/rfc/rfc4716.txt>
- [4] <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- [5] OpenSSH under Linux cannot read ppk files generated by PuTTY under Windows. When you got commit access granted and want to use the `ssh-add` command to add your private key as identity, and when you see "Bad passphrase, try again for /path/my.key:" then you may need to reload your ppk key using PuTTYgen and export it as *OpenSSH* key:
PuTTYgen → Conversions → Export OpenSSH key (source (<http://www.linuxforen.de/forums/showpost.php?p=1381584&postcount=5>)
- [6] "Dealing with Private Keys in Other Formats" (http://winscp.net/eng/docs/ui_puttygen) - article about the different private key formats of PuTTY, OpenSSH, and ssh.com and how to convert between them
- [7] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-August/033179.html>
- [8] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-September/026331.html>
- [9] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-August/026086.html>
- [10] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-March/030151.html>
- [11] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/029690.html>
- [12] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/029640.html>
- [13] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/029651.html>
- [14] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-February/029652.html>
- [15] <http://lists.wikimedia.org/pipermail/wikitech-l/2007-March/030298.html>
- [16] <http://lists.wikimedia.org/pipermail/wikitech-l/2006-December/028158.html>
- [17] <http://thread.gmane.org/gmane.science.linguistics.wikipedia.technical/48847/focus=48869>
- [18] <irc://irc.freenode.net/mediawiki>
- [19] <http://bugzilla.wikimedia.org/>
- [20] <http://mail.wikipedia.org/pipermail/wikitech-l/>
- [21] <http://mail.wikipedia.org/pipermail/mediawiki-cvs/>
- [22] <http://svnbook.red-bean.com/en/1.6/svn.branchmerge.basicmerging.html#svn.branchmerge.basicmerging.undo>
- [23] <http://svn.wikimedia.org/users.php>

Further reading

- SVN Book (<http://svnbook.red-bean.com/en/1.6/index.html>) – a comprehensive help resource for version control with subversion

Development policy

The MediaWiki software is developed collaboratively by hackers from all around the world. To make sure that none of our dozens of wikis breaks by an upgrade, **but to** also make sure that all our wikis are reasonably up-to-date, we need a general **development policy**. This is outlined here below.

Subversion access

Our current policy for handing out Subversion access is very liberal. If you have made meaningful contributions (by submitting patches, preferably as attachments to bugs in our Bugzilla tracker ^[1]), you can apply for Subversion access (see Commit access requests), and it will be granted in most cases. If you just intend to commit and maintain an extension you've written, Subversion access can be granted even without prior participation in development. Subversion is a lot like wiki: Everything can be reverted. On the other hand, Subversion should be kept in a consistent state, especially the stable version (see below).

Unstable, the test server, and the REL* branches

Main development takes place in the main trunk of Subversion, it is unstable, however should always be kept in a working state. Major modifications should take place in a development branch or elsewhere.

Code is generally synced from Subversion main trunk with the Test Wikipedia ^[2] before it goes live on production sites, this is to ensure that everything isn't taken out by the update. It is important to remember that this is not a substitute for testing and all code should thoroughly be tested before being checked into Subversion trunk.

If you care about release management or want to try out one of the stabilized versions, the only branches that matter are those of the form REL*. All others, including "stable", are merely kept around to preserve the Subversion history.

Typically there will be a testing phase when a new release is begun. At that point users will be pointed to test.wikipedia.org to try out all the new things. There might be a few release candidates, a 1.x.0 version, and finally a few minor bugfix releases.

Releases

Every few months, the stable branch should be packaged into a .tar.gz file and made accessible from the webserver. This ensures that users of our software can easily get a reasonably stable version without messing with Subversion. Anyone submitting patches should preferably use the latest version of trunk to avoid conflicts, and of course people with commit access need to use Subversion to be able to commit stuff. It's not advisable to run production wikis off of development versions unless you know what you're doing (as do, for instance, the Wikimedia tech staff, a number of whom are also MediaWiki developers).

The Release checklist provides a more comprehensive list of tasks to be done for a release.

Committing to Subversion unstable

Even the unstable version should at least be *runnable* at any given time; that is, no scripts should throw fatal errors. Minor bugs can be worked out over time, but please test everything locally before committing. Also, while committing the unstable version, you should mention that your code needs fixes, this will help in reviewing it.

Database patches

If the database schema is changed, you need to update `maintenance/updaters.inc` and add an appropriate SQL update file to `maintenance/archives/`. Look at the commit history of `updaters.inc` to find an example of how it's done. *All mandatory schema changes need to be approved by the sysadmins first.* Wikipedia is regularly synchronized to trunk (scapped), and it takes considerable planning to apply schema changes to MySQL-based sites the size of Wikipedia. When new changes are being reviewed prior to scapping, if the reviewer sees an unexpected schema change, he's likely to just revert it rather than delaying the scap for that one commit.

As of 1.17 this is out-of-date: it's in `includes/installer/(MySQL|Postgres|SQLite)Updater.php`

If you want to commit a schema change that can't be very easily applied (things like table creation are usually okay), either get explicit approval from the sysadmins; or make sure it's optional, i.e., that nothing will break if it's not applied right away. You could set your new feature to only work if a config option is set to true, for instance, and set the option to false by default. Then the commit can be safely scapped before the schema change is made. You'll then have to pester the sysadmins incessantly to get them to make the schema change, at which point you can remove the config option to enable your feature.

Test server policy

There is an official test server at <http://test.wikipedia.org>. Some developers may also have test servers deployed.

Generally, the purpose of such "fresh from Subversion" servers is to test the latest unstable branch of the code, *not* to have a testbed for demonstrating unfinished features. Features should be demonstrated with GUI mock-ups before they are implemented (these can be uploaded to Wikimedia Commons (preferred) or to this wiki), and implementations should only be committed to Subversion if they are in a reasonably complete state. This ensures that we avoid features that are never finished. To test your own code, please make sure you have a fully working local MediaWiki installation; if you want *others* to test your unstable code, you can always set up your own test server.

Documentation policy

If a feature is going to be visible to end users or administrators, it's nice if the developer documents the feature on this wiki. Just a few sentences are fine: what it does, how to use it. Others will fix your prose up -- the important part is to have features documented. There are various templates available on this wiki using which you just have to edit a few lines & the template will handle everything else. If you're too lazy, on the other hand, it seems that plenty of people are willing to do it for you (yay wikis).

References

[1] <http://bugzilla.wikimedia.org>

[2] <http://test.wikipedia.org>

USERINFO file

Each MediaWiki developer has a **USERINFO file** in the MediaWiki Subversion repository which gives a bit of extra information about them. As well as being viewable from within the SVN repository (in the `/USERINFO/`^[1] folder) these files are used to generate the list of SVN users located at <http://svn.wikimedia.org/users.php>.

It is the responsibility of all SVN users to create and maintain their own USERINFO file.

Creating a USERINFO file

When you are granted SVN commit access, your first action should be to create your USERINFO file, as described in this section.

***Note:** Instructions are for the SVN command-line client - if you use an alternative client (e.g. TortoiseSVN) then the exact details may differ, but the steps remain the same.*

This example assumes an SVN username of `mysterywizard`:

1. Checkout the USERINFO folder to your local machine:

```
svn checkout svn+ssh://mysterywizard@svn.wikimedia.org/svnroot/mediawiki/USERINFO/ USERINFO
```

2. Change directory to the USERINFO directory you have just created.

```
cd USERINFO
```

3. Create a new file and fill in your details (see below). The file name should be identical to your SVN user name (in this example, the file would simply be called `mysterywizard`).
4. Add the new file to SVN:

```
svn add mysterywizard
```

5. Make sure the eol-style is set on this new file (see also Subversion/auto-props)

```
svn propset svn:eol-style native mysterywizard
```

Note: Set this manually even if you've already setup auto-props. The USERINFO file has no extension and hence no defaults.

6. Commit your changes to the repository:

```
svn commit --message="Adding my USERINFO file."
```

If the commit was successful, then you're done. You can now delete the USERINFO directory that you checked out.

Contents of the USERINFO file

The contents of the file should use the following template:

```
name: Alan Smithee
email: mysterywiz@example.com
irc: asmith
url: http://www.mediawiki.org/wiki/User:MysteryWizard
project: Core JavaScript
since: 20100613
languages spoken: English, Dutch
```

The URL should ideally be to your user account on this wiki, but may be to your user account on another WMF wiki or even to your own website, if you feel that is more appropriate.

If you have previously committed to SVN under a different user name then you should also add an `aliases` line, as described in the next section.

fields description

At the minimum your profile should contains the following fields :

`name`

real name

`email`

your email address which may be obfuscated

`url`

your user page, website, blog or personal homepage (one valid url, otherwise empty)

`since`

the date you have been granted commit rights (format YYYYMMDD)

`project`

extensions you are working on, 'core' if you have commit rights to MediaWiki trunk, branches you maintain, language, 'pywikipedia' for the python wikipedia framework. Fields separated by comma (,).

`languages spoken`

any languages you understand. Might be useful for mail / chat and general communication among developers.

Some fields are optional and should be filled only when they differs from your username:

`irc`

nickname you are using on freenode if it differs from your name

`aliases`

see below

We are liberal about those fields, so feel free to add more of them. Below is a list of well known additional fields:

`wiki`

URL to your main userpage on one of the Wikimedia projects

`twitter`

your twitter account

`identica`

your identi.ca account

Changes to your user name

If your SVN user name changes then you should rename your USERINFO file appropriately, using an `svn mv` operation.

You should also add an `aliases` line to your USERINFO file (as part of the same commit) linking to your old user name:

```
aliases: myoldname
```

If you have had multiple previous IDs, separate them using commas.

References

[1] <http://svn.wikimedia.org/viewvc/mediawiki/USERINFO/>

Subversion

shortcut SVN

MediaWiki uses the Subversion^[1] version control system. To use it, you have to download the official command line Subversion client^[1]. You can also use alternative clients, such as the graphical TortoiseSVN^[2] for Windows.

Subversion's command-line interface is similar to Concurrent Versions System (CVS^[3]). This is a **basic** and partial guide of the most useful commands. For a complete guide, use the book Version Control with Subversion^[4]. If you have write access to the server, it's recommended that you read this book and learn the advanced features of Subversion.

The repository

The MediaWiki repository is hosted by the Wikimedia Foundation and is reachable from <http://svn.wikimedia.org/svnroot/mediawiki/>^[5]. The project uses a more or less standard hierarchy for Subversion repositories, described below as of november 2011:

- `USERINFO`
- `branches`
- `tags`
- `trunk`
- `wikimedia-web`

Pre-production work is made in the `trunk` tree. Wikimedia servers used to run the code in this tree for production usage, as of November 2011, it is not the case but we expect nonetheless a clean trunk. This mean that patches made in this tree **must not break the code**. You should also avoid rewriting extensive portions of the trunk as well. MediaWiki itself is in the **phase3** subdirectory of the trunk.

`branches` is used for major coding work, testing huge patches, and testing unstable patches. Some developers have their own branch of the code here. It is also used to prepare new stable releases (through the well-known beta → release candidate → security & maintenance fix cycle). Each major stable series gets its own directory as `REL<major_version>_<minor_version>`. For example, MediaWiki 1.18 work is being done in `REL1_18`. Wikimedia servers runs from a branch in `/branches/wmf`. Patches made to a release branch (ex: `REL1_18`) are back ported to the current running `wmf` branch by staff.

`tags` is a special hierarchy used to save the state of the software at a given point in the time. You should NEVER make any change to it as this is only useful when releasing new versions. That task is handled solely by the current MediaWiki release manager (Tim Starling, Sam Reed).

There is an additional subproject named `wikimedia-web`, which hosts the Wikimedia portal files located at `http://www.wikimedia.org/`. You usually don't want to modify anything there without referring to the Wikimedia Foundation and/or the Wikimedia system administrators.

Finally the `USERINFO` project host informations about almost all developers having commit access.

Check out

First, you have to check out the code of MediaWiki. Use the following syntax:

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/folders_to_download sub_folder_name
```

You can browse the code structure using the web interface ^[6] (ViewVC ^[7]). Use the three folders there for different purposes:

- Trunk ^[8] is the main development branch.
- The branches ^[9] are used for stable versions of the core and for versions of extensions that work with those stable versions; and for the development of complex features.
- The tags ^[10] are used to track the released versions.

The URL structure is:

transport

`http://svn.wikimedia.org`

repository

`/svnroot/mediawiki`

branch/tag

`/trunk, or /branches/REL1_17, or /tags/REL1_16_2`

files

`/phase3`

Unlike the old CVS, the URL is used to specify the branch or the tag.

Don't leave off the files part (eg. "phase3") or branch part (eg. "trunk")! If you leave that off you check out every revision of every file in the repo, which is insane.

To check out MediaWiki development trunk into the folder "wiki":

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 wiki
```

To check out Extension:TitleKey set into the folder "TitleKey":

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/trunk/extensions/TitleKey TitleKey
```

To check out all extensions set into the folder "extensions":

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/trunk/extensions extensions
```

To check out the latest bits in some particular release branch:

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/branches/REL1_16/phase3 REL1_16
```

To check out a specific version of the software:

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/tags/REL1_16_2/phase3 REL1_16_2
```

To check out a specific development revision number ##### of the software: (for example the one used there: 1.7alpha)


```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3@##### wiki
```

or

```
svn checkout -r ##### http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 wiki
```

or, if you want to switch to that revision #####, use update command:

```
svn update -r #####
```

Unlike the old CVS, anonymous access is completely up to date, so you can grab fixes immediately after they get committed.

Updating the working copy

To update your working copy and get the latest files, use the following command:

```
svn update #or just: "svn up"
```

Note that SVN, unlike CVS, doesn't need to be told to prune removed files or create new directories. This is automagic.

For a simple way to keep abreast of changes, one could use e.g.,

```
cp RELEASE-NOTES /tmp && svn up && diff /tmp RELEASE-NOTES
```

What happens if I change some file, then do "svn up" and it was changed upstream too? Don't worry. You will see

```
Conflict discovered in 'includes/NeurDBall.php'.
```

```
Select: (p) postpone, (df) diff-full, (e) edit, (h) help for more options:
```

Making a diff

Diffs, or patches, are text files which include all the changes done in the working copy. If you suggest a new feature in Bugzilla and like to suggest a change which fixes it, upload a patch.

To create a diff from the current repository, use the following command:

```
svn diff
```

Normally, unlike CVS, you don't have to tell SVN which files you changed; however, you may like to diff only a part of the repository. To do that, specify the files to diff:

```
svn diff includes/SpecialMyAwesomePage.php
```

Note that SVN defaults to the "unified" diff format, so the "-u" option doesn't have to be passed.

You can see the changes that were made since your last checkout:

```
svn diff -rBASE:HEAD
```

or make a forward comparison between your working copy and the latest:

```
svn diff -rHEAD
```

Applying a diff

Subversion does not contain a built in command to apply diffs to the current working copy (for example, to review or commit diffs published in Bugzilla); instead, you can use the regular patch unix utility:

```
patch -p0 < patch_file
```

TortoiseSVN has a built-in support for applying a diff.

Changing file structure

You can add files or folders to the working copy, to be included in the next diff or commit, using the command:

```
svn add file.name
```

If you add a *folder*, all the files included in the folder are added, except for files in the ignored list.

Make sure the eol-style is set on this new file (see also Subversion/auto-props). You may need to set this manually when the file has no or an unusual extension which is not listed in your auto-props setup.

```
svn propset svn:eol-style native file.name
```

You can delete files or folders from the working copy, to be deleted in the next commit or marked as such in the next diff, using the command (which will automatically **delete** the files from the working copy, but won't delete folders in such way):

```
svn delete file.name
```

Make sure the file or folder do not have local modifications, else they won't be deleted unless you force the deletion.

Reverting your changes in your local copy

If you are developer see section Undoing changes below. This is most likely the command you want, and not "svn revert" as explained below.

If your changes in the working copy are not useful in your opinion, you can revert them using the following command:

```
svn revert
```

You must use parameters for this command. To revert all your changes in the working copy, use:

```
svn revert -R .
```

To revert the changes in a specific file, use:

```
svn revert file.name
```

Reverting can also remove added files (they won't be deleted, just removed and considered "unknown files", just like you didn't use `svn add` at first), and restore deleted files (both deleted by hand and deleted by `svn delete`).

Checking the status of the working copy

You can check the status of your working copy using the following command:

```
svn status
```

These are several important letters in the first column of the item, which show the status:

- M = the item was modified by you
- A = the item was added by you (using `svn add`)
- D = the item was deleted by you (using `svn delete`)
- ? = the item is not under the version control, but exists
- != the item is missing (under the version control, but does not exist - probably deleted without using `svn delete`) or incomplete

Developer use

You can request access at the commit access requests page. If you have a write access for the server, you can use an SSH access instead of HTTP access.^[11]

Note: Note that in order for check and commit as developer you must have added your identity to the ssh authentication agent.

For example, if your secret private key is presently located at `/home/tux/.ssh/tux.privatekey`, you must add it using: `ssh-add /home/tux/.ssh/tux.privatekey`

Note: If `ssh-add` says "Could not open a connection to your authentication agent.", then your shell does not run under `ssh-agent`. Type

```
ssh-agent bash[12]  
ssh-add /home/tux/.ssh/tux.privatekey
```

to run `ssh-add` again. (Source ^[13])

URLs

Replace the protocol `http://` with `svn+ssh://` in all the commands (e.g. `svn checkout`) for a developer checkout.

For example, to check out the latest trunk as an anonymous, you use:

```
svn checkout http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 wiki
```

To check it out as a developer, use:

```
svn checkout svn+ssh://username@svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 wiki
```

Passphrase prompts may trigger for authentication -- if you haven't set up a SSH agent to help you with this (many operating systems do this as a standard component) you may wish to consult your system documentation.

If your SVN username does not match your local computer username, you can usually set this **once** in your SSH config file for the host 'svn.wikimedia.org' and will never have to worry about specifying it ever again. You can also embed your username into the URL, like 'svn+ssh://username@svn.wikimedia.org' however this tends to be inconvenient.

SSH will prompt you to verify the SSH key fingerprint for `svn.wikimedia.org`. The correct fingerprint is `4d:76:a4:a2:47:c1:bc:a8:d5:d7:51:ec:15:71:77:9a`.

Recommended checkout

When developing, you will probably use some extensions but will not be willing to actually checkout all extensions. That could slow down your workflow when issuing svn commands at the root of your working copy. The scheme below is the recommended way to set up your local copy:

```
~$ svn co svn+ssh://USERNAME@svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 wiki
// Subversion now fetch MediaWiki in the 'wiki directory
$ cd wiki
// We will now delete the stub directory for extensions:
~/wiki$ rm -fR extensions
// Check out only the root of the extensions:
~/wiki$ svn co svn+ssh://USERNAME@svn.wikimedia.org/svnroot/mediawiki/trunk/extensions --depth empty
U   extensions
Checked out revision 123456.
~/wiki$
```

This way you have not checked out any extension and any subversion command at the root of your repository will not have to look at all extensions:

```
~/wiki$ svn status
      S   extensions
~/wiki$
```

To actually fetch the WikiLove extension:

```
~/wiki$ cd extensions
~/wiki/extensions$ svn update WikiLove
A     WikiLove
// more files being added
Updated to revision 123456.
~/wiki/extensions$
```

Notice we used the 'update' command to fetch the extension. The checkout has already been done previously but with --depth empty. Thus you will not receive any information about extensions you did not request.

"Commit failed" ? Convert your anonymous checkout for developer's use !

If you sent an svn commit ... and your system annoys you with a message like

```
svn: Commit failed (details follow):
svn: Server sent unexpected return value (403 Forbidden) in response to MKACTIVITY request
for '/svnroot/mediawiki/!svn/act/cf44a626-c784-11e0-b670-c722664463a3'
```

you have most likely tried to commit your updated local working copy based on a previous anonymous check-out via http://.

You are lucky: this can be *healed*. For example, if you accidentally checked out your phase3 using http:// instead of the developer checkout svn+ssh://

```
cd /your/local/working/copy/directory
svn switch --relocate \
    http://svn.wikimedia.org/svnroot/mediawiki/ \
    svn+ssh://svn.wikimedia.org/svnroot/mediawiki/
```

Windows users

Windows users of subversion client may need to follow the instructions mentioned here ^[14] to make the tool use their SSH public key (or if you are using TortoiseSVN, checkout this how-to guide ^[15]).

Or it can be loaded into Pageant, and will automatically be used from there. With TortoiseSVN 1.7, make sure to use a current version of pageant (0.61+).

Auto properties

See Subversion/auto-props for how to enable automatic line-ending conversion for files you add. Every developer should use it.

Commits

Commits, or check ins, are the action of applying your changes from the working copy to the web repository. Use the following command to do that:

```
svn commit
```

Using the command without the parameters will fail, unless you've configured an editor, because you have to enter a comment for the file logs. You can use one of the following forms:

```
svn commit --message="This is the log comment for revision r1234 . It fixes bug12345 and introduces \SwgNewVariable."  
svn commit -m "This is the log comment for revision r1234 . It fixes bug12345 and introduces \SwgNewVariable."  
svn commit --file=file_with_log_comment
```

Note: If you need shell-reserved characters like '\$' in your comment, escape them with a backslash as shown.

To cause an extension to show up under the correct version in Special:ExtensionDistributor, add your extension to the corresponding extensions branch in /branches/, for example /branches/REL1_15/extensions/ParserFunctions/.

Magic linking

Syntax like

- bug 12345
- r12345
- follow up patch to r12345

in a commit message or code comment automatically creates a link to the specific bug or revision, or set a follow-up indication at the specific revision it follows up. When used in a commit message specifically, this syntax also creates links between revisions (to note follow-up revisions, for example).

Undoing changes

If you commit a change, and it turns out to be completely broken, then you can revert it by doing something like this (undoing changes explained in detail ^[16]).



Warning: Before performing a merge, you should always use it with option `--dry-run` as in the example

```
svn merge -c -89151 --dry-run .
```

to preview changes and potential conflicts; no changes are applied to the working copy or the repository. ^[17]

```
svn up  
svn merge -r REVISION_NUM_YOU_WANT_TO_UNDO:REVISION_NUM_TO_REVERT_TO .
```

```
svn status
svn diff
svn commit --message="Self-revert accidental breakage" includes/file-you-changed.php includes/another-file-you-changed.php
```

For reverting a single version, you can use the changeset notation (take note of the - sign before the revision number to apply the reverse difference)

```
svn up
svn merge -c -REVISION_YOU_WANT_TO_UNDO .
svn status
svn diff
svn commit --message="Self-revert accidental breakage" includes/file-you-changed.php includes/another-file-you-changed.php
```

Using the GitHub mirror

There's a mirror of the MediaWiki SVN repository at GitHub ^[18]. To use it:

First clone the repository:

```
git svn clone git://github.com/mediawiki/mediawiki-svn.git
```

This will take a while. The repository is around 1.15 GiB (as of November 5, 2011).

Committing back from this mirror is currently unsupported. See this thread ^[19] on the Git mailing list for why. The gist of it is that it's hard to reconstruct the SVN metadata needed to push back to the original SVN.

What you can do instead is this:

Check out a limited part of the repository instead of the whole thing, right now there are trunk/phase3 and trunk/extensions mirrors:

```
git clone git://github.com/mediawiki/mediawiki-trunk-phase3.git
```

or:

```
git clone git://github.com/mediawiki/mediawiki-trunk-extensions.git
```

Hack there on your own branch and keep syncing with upstream. Then when you're ready to push back create your own mirror (or ask avar for an up to date copy):

```
git svn clone svn+ssh://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3
```

Then just move your commits between the two repositories. E.g. with `git format-patch` in the original followed by `git am` in the other.

Converting a CVS checkout to SVN

Assuming you don't want to keep any local changes to files in the repository, it's easy to just overwrite everything with a fresh checkout. This will keep your local files, such as LocalSettings.php and custom skins.

```
svn co http://svn.wikimedia.org/svnroot/mediawiki/trunk/phase3 temp-checkout
rsync -a temp-checkout/ /path/to/phase3/
```

The following works if you didn't delete any directories:

```
svn revert -R /path/to/phase3
```

And if you want to get rid of the old CVS dirs:

```
find . -type d -name CVS -print0 | xargs -0r rm -rf
```

Be careful with that one. ;)

If you don't have SVN or can't use it

At <http://toolserver.org/~vvv/mw-nightly/> you can find snapshots of trunk and extensions, updated daily.

External links

- Subversion Web access ^[6]
- Useful SVN commands ^[20] - a longer and more detailed overview (Greenstone wiki)
- Version Control with Subversion ^[21] book (SVN version 1.4)
- OrganicDesign:MediaWiki SVN Statistics - statistics about the number of committers and commits to the MediaWiki SVN repository (updated 2010-06-15)

References

- [1] <http://subversion.apache.org/>
- [2] <http://tortoisesvn.tigris.org/>
- [3] <http://www.nongnu.org/cvs/>
- [4] <http://svnbook.red-bean.com/en/1.2/index.html>
- [5] <http://svn.wikimedia.org/svnroot/mediawiki/>
- [6] <http://svn.wikimedia.org/viewvc/mediawiki>
- [7] <http://www.viewvc.org/>
- [8] <http://svn.wikimedia.org/viewvc/mediawiki/trunk>
- [9] <http://svn.wikimedia.org/viewvc/mediawiki/branches>
- [10] <http://svn.wikimedia.org/viewvc/mediawiki/tags>
- [11] This might change later.
- [12] If you use PuTTY for connecting to your server, you can configure PuTTY so that it switches to a directory of your choice and starts such a shell automatically:
 - PuTTY → Connection → SSH → Data to send to the server → Remote command
`cd /work/your-mediawiki-working-directory && ssh-agent bash -login`
- [13] <https://www.cs.indiana.edu/Facilities/FAQ/Security/openssh.html>
- [14] http://guide.apisnetworks.com/index.php/Subversion#Official_svn_client
- [15] http://tortoisesvn.net/ssh_howto.html
- [16] <http://svnbook.red-bean.com/en/1.5/svn.branchmerge.basicmerging.html#svn.branchmerge.basicmerging.undo>
- [17] If you see from a `my_projectfolder# svn merge -c -89151 --dry-run .---` Reverse-merging r89151 into '': C UserMerge.php Summary of conflicts: Text conflicts: 1 or similar messages you can not blindly revert this specific revision because further changes in intermediate revisions have been applied to a text line, so that more things in this line need to be reverted or fixed.
- [18] <http://github.com/mediawiki>
- [19] <http://comments.gmane.org/gmane.comp.version-control.git/146669>
- [20] http://wiki.greenstone.org/wiki/index.php/Useful_SVN_Commands
- [21] <http://svnbook.red-bean.com/en/1.4/index.html>

Language:	English
-----------	---------

Article Sources and Contributors

Coding conventions *Source:* <http://www.mediawiki.org/w/index.php?oldid=461770> *Contributors:* Alphos, Bdk, Brion VIBBER, Bryan, Catrope, Cneubauer, Courant, Dantman, Danwe, Grunny, Happy-melon, HappyDog, Hashar, Helder.wiki, Hoo man, Huji, IALex, Jack Phoenix, Jeroen De Dauw, Jidanni, JonathanWilliford, Jrhizor, Kaldari, Krinkle, MZMcBride, MarkAHershberger, Matěj Grabovský, MaxSem, Mjec, Nickj, Patrick, Peachey88, Ravichandratbv, Reedy, RobLa-WMF, Robchurch, Rogerhc, Sanbeg, Saper, Schnark, Simetrical, Skizzerz, Stéphane Thibault, Tgr, Tim Starling, Tisane, Turnstep, Vincelondon, Wikinaut, X!, Zakgreant, ^demon, Ævar Arnfjörð Bjarmason, 17 anonymous edits

JavaScript performance *Source:* <http://www.mediawiki.org/w/index.php?oldid=461769> *Contributors:* Brion VIBBER, Catrope, Grunny, Helder.wiki, IALex, Krinkle, Simetrical, 1 anonymous edits

jQuery *Source:* <http://www.mediawiki.org/w/index.php?oldid=441881> *Contributors:* IALex, Jack Phoenix, Krinkle, Locos epraix, MZMcBride, MaxSem, 2 anonymous edits

Pre-commit checklist *Source:* <http://www.mediawiki.org/w/index.php?oldid=458981> *Contributors:* AJim, Algevis, Eloquence, Grunny, Johnduhart, Krinkle, MarkAHershberger, MaxSem, Mormegil, Purodha, Reach Out to the Truth, Simetrical, Zakgreant, ^demon, 5 anonymous edits

How to debug *Source:* <http://www.mediawiki.org/w/index.php?oldid=461168> *Contributors:* Akmed13, Cneubauer, Courant, Duesentrieb, Eep, Egingell, EricMyers, Gcdinsmore, Hashar, IALex, Jack Phoenix, Jeremyb, Matěj Grabovský, MaxSem, Mediashrek, Michael Daly, Nad, Nikerabbit, Pathoschild, Reedy, Rolandp, Ryanbobko, Skierpage, TheFearow, Thomas Klein, Tisane, Trevor Parscal, Werdna, X!, Zven, 16 anonymous edits

Requests for comment page: discuss your new idea before committing *Source:* <http://www.mediawiki.org/w/index.php?oldid=458424> *Contributors:* Brion VIBBER, Dantman, IALex, Jarry1250, Kaldari, Krinkle, MZMcBride, Olivier Beaton, Platonides, RobLa, RobLa-WMF, Siebrand, Sumanah, Wikinaut, ^demon, 1 anonymous edits

Code review *Source:* <http://www.mediawiki.org/w/index.php?oldid=448132> *Contributors:* Jack Phoenix, Krinkle, Luckas Blade, MZMcBride, Nemo bis, Reedy, RobLa-WMF, Siebrand, Tisane, Wikinaut, Zakgreant, ^demon, 3 anonymous edits

Code review guide *Source:* <http://www.mediawiki.org/w/index.php?oldid=453124> *Contributors:* Bawolff, Bryan, Guillom, Hashar, MZMcBride, Peachey88, Sumanah, Tim Starling, Wikinaut, Zakgreant, ^demon

Code review tags *Source:* <http://www.mediawiki.org/w/index.php?oldid=436485> *Contributors:* Krinkle, MarkAHershberger, Peachey88, RobLa-WMF, Sumanah

Security *Source:* <http://www.mediawiki.org/w/index.php?oldid=429554> *Contributors:* Ans, Brion VIBBER, Courant, Egfrank, HappyDog, IALex, Jcpren, Liangent, MZMcBride, Stéphane Thibault, Tim Starling, 3 anonymous edits

Security for developers *Source:* <http://www.mediawiki.org/w/index.php?oldid=454773> *Contributors:* BrianOregon, Dcx, Emufarmers, Happy-melon, Hydриз, Krinkle, MarkAHershberger, Matěj Grabovský, MaxSem, Peachey88, Platonides, RobLa-WMF, Tim Starling, Wikinaut, Zakgreant, 9 anonymous edits

Security (Manual) *Source:* <http://www.mediawiki.org/w/index.php?oldid=460592> *Contributors:* AaronPeterson, Achimbode, Amgine, Asb, Bdk, Bookofjude, Brion VIBBER, Bryan, Chira, Choshii, Cspurrier, Ed, Edward Chernenko, Emufarmers, Gmoon, Happy-melon, HappyDog, IALex, Jean-Christophe Chazalotte, Jidanni, Korg, Korrigan, Lazer erazer, Liangent, Lowellian, MaxSem, Mga, Mrgoblin, Nilsk, Omniplex, Plugwash, Rene Pijlman, Spacebirdy, Tim Starling, Timneu22, Umasstrower, Vxbush, Wikinaut, Wutsje, Zakgreant, ^demon, 78 anonymous edits

Cross-site scripting *Source:* <http://www.mediawiki.org/w/index.php?oldid=454774> *Contributors:* Dantman, Dcx, Kaldari, Peachey88, Wargo, Zakgreant

Cross-site request forgery *Source:* <http://www.mediawiki.org/w/index.php?oldid=423058> *Contributors:* Peachey88, Wikinaut, Zakgreant, 1 anonymous edits

Register globals *Source:* <http://www.mediawiki.org/w/index.php?oldid=461397> *Contributors:* Hoo man, MaxSem, Ucucha, Wargo, Zakgreant

SQL injection *Source:* <http://www.mediawiki.org/w/index.php?oldid=422122> *Contributors:* IALex, Wargo, Zakgreant, ^demon

Database access *Source:* <http://www.mediawiki.org/w/index.php?oldid=440161> *Contributors:* Alvin-cs, Bdk, Bryan, Courant, Dmb, Eep, Errectstapler, Gigs, IALex, Jack Phoenix, Kappa, Lampak, MaxSem, Nad, Olenz, OverlordQ, Patrick, Peachey88, Sledged, Tisane, Zakgreant, ^demon, 6 anonymous edits

Securing database passwords *Source:* <http://www.mediawiki.org/w/index.php?oldid=392502> *Contributors:* AaronPeterson, Alrik, Aotake, Cthomas, Dark Mage, Feydey, Happy-melon, IALex, JT, Kghbln, Lenhan, MaxSem, Michael Daly2, Msfz751, Natmaka, Omniplex, Plugwash, Reach Out to the Truth, Vt-aoe, 35 anonymous edits

Extensions *Source:* <http://www.mediawiki.org/w/index.php?oldid=460412> *Contributors:* AlisonW, Bawolff, Bdk, Bencmq, Bn2vs, Bouncingmolar, BrianOregon, Btilm, Catrope, Cneubauer, Courant, Cumeo89, Dantman, Dmb, Duesentrieb, Eep, Egfrank, Flominator, Gpkarthik, Harry Wood, Hoggwild5, Hoo man, Hopefully acceptable username, Hsilamot, IALex, Iste Praetor, JenVan, JiaoyanChen, Kaganer, Karl Cheung, Kghbln, MC10, Mange01, Matanya, Mathonius, MaxSem, Melissa 4.0, Moejoe000, Mr.Z-man, NicoV, Nikerabbit, Pannini, Patrick, Peachey88, Purodha, Reedy, Revolus, SPQRRobin, Samba, Sumanah, TBloemink, Tbleher, TheFearow, Titoxd, Varnent, Vigilius, Vikici, Voyagerfan5761, Willsrlrt, Zakgreant, Zven, 43 anonymous edits

Developing extensions *Source:* <http://www.mediawiki.org/w/index.php?oldid=461349> *Contributors:* Badon, Bawolff, Dmb, Grunny, Hoo man, IvanLanin, Juju2004, MaxSem, Mickeyf, Olivier Beaton, Peachey88, Sumanah, Varnent, Wargo, Wikinaut, Zakgreant, 8 anonymous edits

Parser functions *Source:* <http://www.mediawiki.org/w/index.php?oldid=460415> *Contributors:* AS, Bawolff, Catrope, Cneubauer, Courant, Csagedy, Dodoïste, Egfrank, Errectstapler, Flominator, G.Hagedorn, Hoo man, JenVan, Jeroen De Dauw, Lil devil, Mr.Z-man, Nx, Patrick, Plustgarten, SPQRRobin, Swift, Tbleher, Tisane, Varnent, Wutsje, Zven, ^demon, 11 anonymous edits

Special pages, OutputPage, WebRequest, Database, User, Title.php, FAQ *Source:* <http://www.mediawiki.org/w/index.php?oldid=460417> *Contributors:* ++ic, Aaron Schulz, Ahyl, Alvin-cs, Anonymous Dissident, Aretai, Bawolff, Behrang Amini, Brion VIBBER, Catrope, Church of emacs, Cmorse, Cneubauer, Cometstyles, Courant, DanielRenfro, Delete, Dgriff, Dto, Edward Z. Yang, Eep, Egfrank, Emufarmers, Fomafix, Gogebic, Grondin, Grunny, GunterS, Happy-melon, HappyDog, Hillgentleman, IALex, IndyGreg, Ivolucien, Jack Phoenix, Jelte, Jibbles, Jidanni, JimHu, Jimbojw, Johnduhart, JonathanWilliford, Kaldari, Kevang, Langec, Laszlo, Liangent, Maiden taiwan, Mandaliet, Mati, MaxEnt, MaxSem, Michael Daly2, Morton, MrBlueSky, Nikerabbit, Patrick, PhilBoswell, RapidRocker, Reach Out to the Truth, Rick DeNatale, Robchurch, SHL, SPQRRobin, ST47, Sawdabee, Sergey Chernyshev, Shanel, Siebrand, Skizzerz, Slanted, Stinkfly, Stéphane Thibault, Subfader, TheFearow, Tim Starling, Tisane, Tkl, Varnent, VittGam, Waihorace, Wikinaut, Wikitonic, Woosle, Wutsje, Zven, 94 anonymous edits

Title.php *Source:* <http://www.mediawiki.org/w/index.php?oldid=440038> *Contributors:* Courant, Dmb, Gogebic, IALex, Wikinaut, Zakgreant, 4 anonymous edits

\$wgLegalTitleChars *Source:* <http://www.mediawiki.org/w/index.php?oldid=440035> *Contributors:* Dmb, Emufarmers, HappyDog, IALex, Pathoschild, Patrick, 4 anonymous edits

Extending wiki markup *Source:* <http://www.mediawiki.org/w/index.php?oldid=370136> *Contributors:* Cabalamat, Cneubauer, Courant, Duesentrieb, Egfrank, Emufarmers, Jeffmcneill, Liangent, Plustgarten, Purodha, Siebrand, 7 anonymous edits

Magic words (Manual) *Source:* <http://www.mediawiki.org/w/index.php?oldid=460419> *Contributors:* Alriode, BRUTE, Bdk, Catrope, Cneubauer, Courant, Eep, Egfrank, Elonka, Goodmorningworld, Hoo man, IALex, Jimbojw, Krinkle, Macv, Nad, Naresnharesh, NigeIJ, Patrick, Savh, TechControl, Timneu22, Varnent, Waldir, 14 anonymous edits

Magic words (Help) *Source:* <http://www.mediawiki.org/w/index.php?oldid=461776> *Contributors:* 333, Al Maghi, Allen4names, Anomie, Anonymous Dissident, Az1568, BRUTE, Bawolff, Bdk, Bilge, Billingham, BrianOregon, CWenger, Catrope, Charitwo, Chris Capocchia, Church of emacs, Ciencia Al Poder, Cimon Avaro, Conrad.Irwin, Courant, Crazymonkey1123, Dmb, Docu, Eep, Egfrank, Elian, Elonka, Emufarmers, Fdsgshdsgsdgh, Foxy Loxy, FrViPofm, Funandtrvl, Graham87, GreenReaper, Gustronico, Hamilton Abreu, Happy-melon, HappyDog, Harry Wood, Hazard-SJ, Hgrosner, IALex, J.delanoy, Jack Phoenix, Jc355h, Jidanni, John Vandenberg, Jordan Brown, Jostar, Jpostlethwaite, Kaganer, Kakurady, Karmela, Kghbln, Korg, Krinkle, Lashuto, Lhrldley, Liangent, Lightsup55, MC10, MZMcBride, MarcoSwart, MathCool10, MaxSem, Mjbauer95, Mr.Z-man, Multi Trixes, Nemo bis, NicDumZ, Nikerabbit, Notortoh, Oculus Spectatoris, OverlordQ, Pascal666, Pathoschild, Patrick, Peteforsyth, PhilPi, Pi zero, Platonides, Rd232, Reach Out to the Truth, Redekopmark, Reedy, Rehman, Reza1615, Rich Farmbrough, Richardguk, Robert Ullmann, Rocket000, Rogerhc, Ronga, Rootover, SPQRRobin, Sanbeg, Saper, Seb35, Shooke, Smith609, Splarka, Srhat, Subfader, Svick, TheObone, Tim Starling, Timeroot, Timeshifter, Tnabtaf, Umherirrender, Voomoo, Waldir, Wikinaut, Woosle, Wutsje, X!, Yarnalogo, Yecril, ZabMilenko, Zigger, ^demon, Ævar Arnfjörð Bjarmason, 45 anonymous edits

Hooks *Source:* <http://www.mediawiki.org/w/index.php?oldid=460766> *Contributors:* 65465798a4654954, ABCD, Adamthec clown, Ajraddatz, BRUTE, Bawolff, Bdk, Bene, BeneM, Bryan, Catrope, Cneubauer, Courant, Dawg, Drakejustin, Drdamour, Duesentrieb, Eep, Egfrank, Emceelam, Fernando.correia, GreenReaper, Grubber, HappyDog, IALex, Jack Phoenix, Jldupont,

Jlerner, Jpond, Krinkle, LeonWeber, Lil devil, Markus Krötzsck, Matěj Grabovský, MaxSem, Mr.Z-man, Nad, Nekocue, Nephele, Nguyenthphuc, Patrick, Phil Boswell, Pinky, Plustgarten, Purodha, Reedy, Rogerhc, Ryan lane, SPQRRobin, Siebrand, Simetrical, Skizzerz, Splarka, Svenir Brkic, Thomas Klein, Tisane, Trevor Parscal, Varnent, VasilievVV, Wikinaut, Wikitonic, Yurik, Zakgreant, Zven, 33 anonymous edits

Extension hook registry *Source:* <http://www.mediawiki.org/w/index.php?oldid=433213> *Contributors:* Anonymous Dissident, Bawolff, Bouncingmolar, Eep, Egfrank, Gri6507, Happy-melon, IALex, Jack Phoenix, JimHu, Jldupont, MarkAHershberger, MaxSem, Nad, Platonides, RV1971, Ryan lane, Stevel, Stéphane Thibault, Svenir Brkic, Wikinaut, Wookienz, 1 anonymous edits

ResourceLoader *Source:* <http://www.mediawiki.org/w/index.php?oldid=457075> *Contributors:* Alolitas, Catrope, Guaka, Guillom, Helder.wiki, Hoo man, Jarry1250, Krinkle, MZMcBride, Mdale, NeilK, Od1n, Onkiro, Rd232, RobLa, RobLa-WMF, Saper, Sumanah, Tim Starling, Trevor Parscal, Zakgreant, 1 anonymous edits

ResourceLoader/Requirements *Source:* <http://www.mediawiki.org/w/index.php?oldid=390731> *Contributors:* Krinkle, Trevor Parscal

ResourceLoader/Migration guide for extension developers *Source:* <http://www.mediawiki.org/w/index.php?oldid=428906> *Contributors:* Krinkle, Nx, Saper, Trevor Parscal, Zakgreant, 1 anonymous edits

ResourceLoader/JavaScript Deprecations *Source:* <http://www.mediawiki.org/w/index.php?oldid=427485> *Contributors:* Bawolff, Brion VIBBER, Catrope, Darklama, DieBuche, Jack Phoenix, Johnduhart, Krinkle, Nihiltres, Nx, Od1n, PDD, Trevor Parscal, UncleDouggie, Zakgreant, 4, אִשְׁרָאֵלִי anonymous edits

Testing portal *Source:* <http://www.mediawiki.org/w/index.php?oldid=453064> *Contributors:* Brion VIBBER, Dmb, Hashar, Krinkle, Leisa, MaxSem, OIEnglish, Olena, Sumanah, VasilievVV, ^demon

Writing testable PHP code *Source:* <http://www.mediawiki.org/w/index.php?oldid=448640> *Contributors:* Hashar, MarkAHershberger, MaxSem, ^demon

PHP unit testing *Source:* <http://www.mediawiki.org/w/index.php?oldid=447334> *Contributors:* Brion VIBBER, Hashar, Jeroen De Dauw, Kaldari, Krinkle, MarkAHershberger, MaxSem, Mdale, Nikerabbit, Sumanah, Trevor Parscal, X!, Zakgreant, 6 anonymous edits

JavaScript unit testing *Source:* <http://www.mediawiki.org/w/index.php?oldid=460206> *Contributors:* Brion VIBBER, Hashar, Krinkle, Reach Out to the Truth, SPQRRobin, Sumanah, 3 anonymous edits

Parser tests *Source:* <http://www.mediawiki.org/w/index.php?oldid=453063> *Contributors:* DieBuche, Kaldari, Peachey88, Sumanah

Style guide *Source:* <http://www.mediawiki.org/w/index.php?oldid=432881> *Contributors:* Drorsnir, Jorm (WMF), MZMcBride, Peachey88, Raindrift

Color selection and accessibility *Source:* <http://www.mediawiki.org/w/index.php?oldid=435135> *Contributors:* Hashar, Jorm (WMF), Waldir

Style guide/Forms *Source:* <http://www.mediawiki.org/w/index.php?oldid=433187> *Contributors:* Jorm (WMF), MZMcBride, Mvuijlst, Parul Vora, Peachey88

Error handling and validation *Source:* <http://www.mediawiki.org/w/index.php?oldid=393503> *Contributors:* Jorm (WMF)

Specialized form elements *Source:* <http://www.mediawiki.org/w/index.php?oldid=413984> *Contributors:* Jorm (WMF)

Assets and gallery *Source:* <http://www.mediawiki.org/w/index.php?oldid=403580> *Contributors:* Jorm (WMF), Peachey88

Translatewiki.net *Source:* <http://www.mediawiki.org/w/index.php?oldid=457058> *Contributors:* Anonymous Dissident, Crt, Doug, Hydriz, IALex, Innv, Nikerabbit, Purodha, Reach Out to the Truth, SPQRRobin, Siebrand, Wargo, 3 anonymous edits

Localisation *Source:* <http://www.mediawiki.org/w/index.php?oldid=461178> *Contributors:* .anaconda, Aaron Schulz, Alexf, Alijsh, Allen4names, Amgine, AndreasJS, Anna L Martin, Awjrichards, BrianOregon, Brion VIBBER, Catrope, Courant, Danny B., Dantman, Db12010, DerHexer, Edward Z. Yang, Eleassar, Farm, GerardM, Get It, Graham87, Guillom, Hamaryns, Hashar, Helder.wiki, Herbythyme, Husky, IALex, IainDavidson, Jack Phoenix, Jean-Christophe Chazallete, Kakurady, Kiensvay, Korg, Krinkle, Liangent, Lloffiw, M7, MZMcBride, Marbles, Meno25, Mulukhiyya, Nemo bis, Nick1915, Nickj, Nikerabbit, Oxcguy3, Patrick, Psy guy, Purodha, Rdsmith4, Red Baron, Reinard, Rhclayto, Roosa, Rotemliss, SPQRRobin, Saper, Schwallex, Shinjiman, Siebrand, Sj, Skalman, Slevinski, Spacebirdy, Stv, Stéphane Thibault, Suisui, Sumanah, Tgr, Thogo, Thunderhead, UV, Ugur Basak, Urhixidur, Werdna, Who, Wikinaut, Willemo, Willserlt, Zigger, Ævar Arnfrjörð Bjarmason, 273 anonymous edits

How to become a MediaWiki hacker *Source:* <http://www.mediawiki.org/w/index.php?oldid=460103> *Contributors:* Ais523, Ajraddatz, Alvin-cs, Angela, Anonymous Dissident, Archivist, Awjrichards, Az1568, Bawolff, Bdk, BraneJ, BrianOregon, Brion VIBBER, Bryan, Bryan Derksen, Bsadowski1, Capmo, ChasWarner, Chris Roy, Collinj, Cphoenix, Daniel Mayer, Darrien, Dmb, Drini, ERIKA MOON, EWLloyd, Edward Z. Yang, Egmontaz, Eloquence, Emufarmers, FireDragon, Folajimi, Greudin, Grunny, HappyDog, Harmeet55, Henna, Herbythyme, Hoo man, Huji, IALex, Icep, Jack Phoenix, Jcarroll, Jdforrester, Jidanni, Jorm (WMF), Jroes, KTC, Kaganer, Kaldari, Korg, Krinkle, Kurt Jansson, M7, MC10, MZMcBride, MarkAHershberger, Matěj Grabovský, MaxSem, Maximillion Pegasus, Mediashrek, Mnh, Mulukhiyya, NSK Nikolaos S. Karastathis, Naconkantari, Nbrouard, Nikerabbit, Oliphant, Omariochasseur, Patrick, Peachey88, Pengo, Petr, Platonides, RHaworth, Reach Out to the Truth, Reedy, Siebrand, Simetrical, Sj, Skierpage, Skizzerz, Splarka, Stephen Gilbert, Suisui, Sumanah, Tbleher, The Utahraptor, The bellman, TheFearow, Tim Starling, Timwi, Tiptopten2000, Tisane, Tom Morris, Turnstep, Waldir, Werdna, Wpedzich, Wutsje, Wyvernoid, X!, Zakgreant, 154 anonymous edits

Commit access *Source:* <http://www.mediawiki.org/w/index.php?oldid=458380> *Contributors:* Alexsh, AmyRose000002, Bawolff, Bdk, Brion VIBBER, Bryan, Catrope, Church of emacs, Courant, DaSch, Emufarmers, Graham87, Guillom, Happy-melon, HappyDog, Huji, IALex, Irancima, Jack Phoenix, Jidanni, Jimbojw, Jorm (WMF), Krinkle, Lcawte, MC10, MaxSem, Mentifisto, Nad, Nickj, Nikerabbit, Oscar, Oysterguitarist, Peachey88, Reedy, RobLa-WMF, Robchurch, RockMFR, Ryan lane, SMcCandlish, SQL, Samsung, Saper, Seb35, Shinjiman, Siebrand, Simetrical, Skizzerz, Stéphane Thibault, Sumanah, Teamviewer, Tim Starling, Tiptoty, Tisane, Titoxd, Turnstep, VasilievVV, Werdna, Wikinaut, Zakgreant, Zven, ^demon, 25 anonymous edits

Development policy *Source:* <http://www.mediawiki.org/w/index.php?oldid=428413> *Contributors:* Akshay.agarwal, Alno, Angela, BrianOregon, Brion VIBBER, Egmontaz, Eloquence, Evan, Funguy77, IALex, IMSoP, Jack Phoenix, Jeronim, Jodi.a.schneider, MaxSem, Mdd4696, Nasa-verve, Purodha, RobertL, Simetrical, Stéphane Thibault, Sumanah, Toby Bartels, Turnstep, Zakgreant, ^demon, 11 anonymous edits

USERINFO file *Source:* <http://www.mediawiki.org/w/index.php?oldid=455650> *Contributors:* HappyDog, Johnduhart, Krinkle, Peachey88, Reedy, Tisane, Yuvipanda, ^demon, 2 anonymous edits

Subversion *Source:* <http://www.mediawiki.org/w/index.php?oldid=457059> *Contributors:* Adamthecrown, Al Maghi, AmiDaniel, Bdk, Brion VIBBER, Catrope, Church of emacs, Courant, Duesentrieb, Eloquence, Graham87, Gurch, HappyDog, Hashar, Huji, Jidanni, Kghbln, Krinkle, MC10, Matěj Grabovský, MaxSem, Mglaser, Nad, Nyks, Pathoschild, Peachey88, Phil Boswell, Platonides, Purodha, Reedy, RobLa, Robchurch, Rotemliss, SColombo, Sanbeg, Shinjiman, Siebrand, Stéphane Thibault, Tim Starling, Timwi, Tisane, Titoxd, Voyagerfan5761, Wikinaut, Wutsje, Zakgreant, ^demon, Ævar Arnfrjörð Bjarmason, 27 anonymous edits

Image Sources, Licenses and Contributors

File:Attention niels epting.svg *Source:* http://www.mediawiki.org/w/index.php?title=File:Attention_niels_epting.svg *License:* Public Domain *Contributors:* niels epting

File:Code review Sign-offs.png *Source:* http://www.mediawiki.org/w/index.php?title=File:Code_review_Sign-offs.png *License:* Creative Commons Zero *Contributors:* User:Wikinaut

File:Code review Sign-offs tested.png *Source:* http://www.mediawiki.org/w/index.php?title=File:Code_review_Sign-offs_tested.png *License:* Creative Commons Zero *Contributors:* User:Wikinaut

File:Crystal Clear app database.png *Source:* http://www.mediawiki.org/w/index.php?title=File:Crystal_Clear_app_database.png *License:* GNU Free Documentation License *Contributors:* Augiasstallputzer, CyberSkull, Herbythyme, 1 anonymous edits

Image:MediaWiki-extensions-icon.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MediaWiki-extensions-icon.svg> *License:* Creative Commons Attribution-Sharealike 3.0 *Contributors:* User:Varnent

File:MediaWiki-extensions-icon.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MediaWiki-extensions-icon.svg> *License:* Creative Commons Attribution-Sharealike 3.0 *Contributors:* User:Varnent

Image:Nuvola apps important.svg *Source:* http://www.mediawiki.org/w/index.php?title=File:Nuvola_apps_important.svg *License:* GNU Lesser General Public License *Contributors:* Bastique

Image:Geographylogo.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:Geographylogo.png> *License:* Public Domain *Contributors:* ALE!, DieBuche, Duesentrieb, Emuzesto, Ephemeron, Multichill, Niki K, Paddy, Pfctdayelise, ~Pyb

File:Low-hanging-fruit-vs-micro-optimization.theora.ogv *Source:* <http://www.mediawiki.org/w/index.php?title=File:Low-hanging-fruit-vs-micro-optimization.theora.ogv> *License:* Creative Commons Attribution-Sharealike 3.0 *Contributors:* Catrope, Peachey88, Sumanah

File:Screenshot-MediaWiki JavaScript Test Suite - Mozilla Firefox.png *Source:* http://www.mediawiki.org/w/index.php?title=File:Screenshot-MediaWiki_JavaScript_Test_Suite_-_Mozilla_Firefox.png *License:* unknown *Contributors:* Brion VIBBER, Krinkle

File:Screenshot-MediaWiki QUnit CompletnessTest.png *Source:* http://www.mediawiki.org/w/index.php?title=File:Screenshot-MediaWiki_QUnit_CompletnessTest.png *License:* unknown *Contributors:* Krinkle

File:MW-StyleGuide-FormExample-CreateAccount.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-FormExample-CreateAccount.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-FormExample-Login.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-FormExample-Login.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Normal.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Normal.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Checkbox-Group.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Checkbox-Group.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Radio-Group.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Radio-Group.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-SelectBox.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-SelectBox.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Checkbox-Single.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Checkbox-Single.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Button-Grouping.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Button-Grouping.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Buttons-Inactive-To-Active.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Buttons-Inactive-To-Active.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Optional.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Optional.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Required.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Required.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Focused.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Focused.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Help.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Help.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Help-Hover.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Help-Hover.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Instructions.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Instructions.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Hint.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Hint.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Input-Hint-Focused.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Input-Hint-Focused.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-AlertMark.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-AlertMark.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-CheckMark.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-CheckMark.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-NoMark.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-NoMark.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-AlertMark.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-AlertMark.svg> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-CheckMark.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-CheckMark.svg> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-NoMark.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-NoMark.svg> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Toggles.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Toggles.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-LiveUpdate-Password-Example.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-LiveUpdate-Password-Example.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-LiveUpdate-UserName-Example.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-LiveUpdate-UserName-Example.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Error-Banner.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Error-Banner.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Error-Banner-List.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Error-Banner-List.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Error-Banner-System.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Error-Banner-System.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Error-Vertical-Style.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Error-Vertical-Style.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Error-Horizontal-Style.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Error-Horizontal-Style.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-CallToAction.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-CallToAction.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Thermometer-Scaled.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Thermometer-Scaled.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-Thermometer-Unscaled.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-Thermometer-Unscaled.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-StarBars.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-StarBars.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-StyleGuide-ProgressMonitor.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-StyleGuide-ProgressMonitor.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-Warning.png *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-Warning.png> *License:* GNU General Public License *Contributors:* Jorm (WMF)

File:MW-Icon-Warning.svg *Source:* <http://www.mediawiki.org/w/index.php?title=File:MW-Icon-Warning.svg> *License:* GNU General Public License *Contributors:* Jorm (WMF)

License

Creative Commons Attribution-Share Alike 3.0 Unported
[//creativecommons.org/licenses/by-sa/3.0/](https://creativecommons.org/licenses/by-sa/3.0/)
