



## **The Handbook**

### **Codex**

[http://codex.wordpress.org/Function Reference/WP Rewrite](http://codex.wordpress.org/Function_Reference/WP_Rewrite)

### **Version Date**

9 August 2005

# Function Reference/WP Rewrite

This document assumes familiarity with [Apache](http://httpd.apache.org/) (<http://httpd.apache.org/>)'s [mod\\_rewrite](http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html) ([http://httpd.apache.org/docs/2.0/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html)). If you've never heard of this before, try reading Sitepoint's [Beginner's Guide to URL Rewriting](http://www.sitepoint.com/article/guide-url-rewriting) (<http://www.sitepoint.com/article/guide-url-rewriting>).

## *Role of WP\_Rewrite*

`WP_Rewrite` is WordPress' class for managing the rewrite rules that allow you to use [Pretty Permalinks](#) feature. It has several methods that generate the rewrite rules from values in the database. It is used internally when updating the rewrite rules, and also to find the URL of a specific post, Page, category archive, etc.. It's defined in `wp-includes/wp-classes.php` and a global [singleton](#), `$wp_rewrite`, is initialised in `wp-settings.php`.

## *Methods and Properties*

This is the formal documentation of `WP_Rewrite`. Try not to access or set the properties directly, instead use the methods to interact with the `$wp_rewrite` object.

### Properties

`$permalink_structure`

The permalink structure as in the database. This is what you set on the Permalink Options page, and includes 'tags' like `%year%`, `%month%` and `%post_id%`.

`$category_base`

Anything to be inserted before category archive URLs. Defaults to `'category/'`.

`$category_structure`

Structure for category archive URLs. This is just the `$category_base` plus `'%category%'`.

`$author_base`

Anything to be inserted before author archive URLs. Defaults to `'author/'`.

`$author_structure`

Structure for author archive URLs. This is just the `$author_base` plus `'%author%'`.

`$feed_base`

Anything to be inserted before feed URLs. Defaults to `'feed/'`.

`$feed_structure`

Structure for feed URLs. This is just the `$feed_base` plus `'%feed%'`.

`$search_base`

Anything to be inserted before searches. Defaults to `'search/'`.

`$search_structure`

Structure for search URLs. This is just the `$search_base` plus `'%search%'`.

`$comments_base`

Anything to be inserted just before the `$feed_structure` to get the latest comments feed. Defaults to `'comments'`.

`$comments_feed_structure`

The structure for the latest comments feed. This is just `$comments_base` plus `$feed_base` plus `'%feed%'`.

`$date_structure`

Structure for dated archive URLs. Tries to be `'%year%/%monthnum%/%day%'`, `'%day%/%monthnum%/%year%'` or `'%monthnum%/%day%/%year%'`, but if none of these are detected in your `$permalink_structure`, defaults to `'%year%/%monthnum%/%day%'`. Various functions use this structure to obtain less specific structures: for example, `get_year_permastruct()` simply removes the `'%monthnum%'` and `'%day%'` tags from `$date_structure`.

`$page_structure`

Structure for Pages. Just `'%pagename%'`.

`$front`

Anything up to the start of the first tag in your `$permalink_structure`.

`$root`

The root of your WordPress install. Prepended to all structures.

`$matches`

Used internally when calculating back references for the redirect part of the rewrite rules.

`$rules`

The rewrite rules. Set when `rewrite_rules()` is called.

`$rewritecode`

An array of all the tags available for the permalink structure. See [Using Permalinks](#) for a list.

`$rewritereplace`

What each tag will be replaced with for the regex part of the rewrite rule. The first element in `$rewritereplace` is the regex for the first element in `$rewritecode`, the second corresponds to the second, and so on.

`$queryreplace`

What each tag will be replaced with in the rewrite part of the rewrite rule. The same correspondance applies here as with `$rewritereplace`.

## Methods

`using_permalinks()`

Returns `true` if your blog is configured to use [Pretty Permalinks](#).

`using_index_permalinks()`

Returns `true` if your blog is using the reduced `mod_rewrite` rules.

`using_mod_rewrite_permalinks`

Returns `true` if your blog is using the extended (verbose) rewrite rule set.

`page_rewrite_rules()`

Returns the set of rules for any Pages you have created.

`get_date_permastruct()`, `get_category_permastruct()`,

`get_date_permastruct()` etc.

Populates the corresponding property (e.g., `$date_structure` for `get_date_permastruct()`) if it's not already set and returns it. The functions `get_month_permastruct()` and `get_year_permastruct()` don't have a corresponding property: they work out the structure by taking the `$date_structure` and removing tags that are more specific than they need (i.e., `get_month_permastruct()` removes the '%day%' tag, as it only needs to specify the year and month).

`add_rewrite_tag($tag, $pattern, $query)`

add an element to the `$rewritecode`, `$rewritereplace` and `$queryreplace` arrays using each parameter respectively. If `$tag` already exists in `$rewritecode`, the existing value will be overwritten.

`generate_rewrite_rules($permalink_structure, $page = true, $feed = true, $forcomments = false, $walk_dirs = true)`

A large function that generates the rewrite rules for a given structure, `$permalink_structure`. If `$page` is true, an extra rewrite rule will be generated for accessing different pages (e.g. `/category/tech/page/2` points to the second page of the 'tech' category archive). If `$feed` is true, extra rewrite rules will be generated for obtaining a feed of the current page, and if `$forcomments` is true, this will be a comment feed. If `$walk_dirs` is true, then a rewrite rule will be generated for each directory of the structure provided, e.g. if you provide it with `'/%year%//%month%//%day%'`, rewrite rules will be generated for `'/%year%/'`, `'/%year%//%month%/'` and `'/%year%//%month%//%day%/'`. This returns an associative array using the regex part of the rewrite rule as the keys and redirect part of the rewrite rule as the value.

`generate_rewrite_rule($permalink_structure, $walk_dirs = false)`

Generates a no-frills rewrite rule from the permalink structure. No rules for extra pages or feeds will be created.

`rewrite_rules()`

populate and return the `$rules` variable with an associative array as in `generate_rewrite_rules()`. This is generated from the post, date, comment, search, category, authors and page structures.

`wp_rewrite_rules()`

returns the array of rewrite rules as in `rewrite_rules()`, but using `$matches[xxx]` in the (where xxx is a number) instead of the normal `mod_rewrite` backreferences, `$xxx` (where xxx is a number). This is useful when you're going to be using the rules inside PHP, rather than writing them out to a `.htaccess` file.

`mod_rewrite_rules()`

returns a string (not an array) of all the rules. They are wrapped in an Apache [`<IfModule>`](http://httpd.apache.org/docs/2.0/mod/core.html#ifmodule) (`http://httpd.apache.org/docs/2.0/mod/core.html#ifmodule`) block, to ensure `mod_rewrite` is enabled.

`set_permalink_structure($permalink_structure)`

Change the permalink structure.

`set_category_base($category_base)`

Change the category base.

`init()`

Set up the object, set `$permalink_structure` and `$category_base` from the database. Set `$root` to `$index` plus `'/'`. Set `$front` to everything up to the start of the first tag in the permalink structure. Unset all other properties.

`WP_Rewrite` (constructor)

Call `init()`.

## Plugin Hooks

As the rewrite rules are a crucial part of your weblog functionality, WordPress allows plugins to hook into the generation process at several points. `rewrite_rules()`, specifically, contains nine filters and one hook for really precise control over the rewrite rules process. Here's what you can filter in `rewrite_rules()`:

- To filter the rewrite rules generated for permalink URLs, use `post_rewrite_rules`.
- To filter the rewrite rules generated for dated archive URLs, use `date_rewrite_rules`.
- To filter the rewrite rules generated for category archive URLs, use `category_rewrite_rules`.
- To filter the rewrite rules generated for search URLs, use `search_rewrite_rules`.
- To filter the rewrite rules generated for the latest comment feed URLs, use `comments_rewrite_rules`.
- To filter the rewrite rules generated for author archive URLs, use `author_rewrite_rules`.
- To filter the rewrite rules generated for your Pages, use `page_rewrite_rules`.
- To filter the rewrite rules generated for the root of your weblog, use `root_rewrite_rules`.
- To filter the whole lot, use `rewrite_rules_array`.
- There is also a hook, `generate_rewrite_rules`. If your function takes a parameter, it will be passed a [reference](http://www.php.net/manual/en/language.references.php) (<http://www.php.net/manual/en/language.references.php>) to the entire `$wp_rewrite` object.

`mod_rewrite_rules()` is the function that takes the array generated by `rewrite_rules()` and actually turns it into a set of rewrite rules for the `.htaccess` file. This function also has a filter, `mod_rewrite_rules`, which will pass functions the string of all the rules, including the `<IfModule>` surrounding section. (Note: you may also see plugins using the `rewrite_rules` hook, but this is [deprecated](#)).

## Example

The most obvious thing a plugin would do with the `$wp_rewrite` object is add its own rewrite rules. This is remarkably simple. Filter the generic `rewrite_rules_array`. The [Jerome's Keywords](http://vapourtrails.ca/wp-keywords) (<http://vapourtrails.ca/wp-keywords>) plugin does this to enable URLs like `http://example.com/tag/sausages`.

```
function keywords_createRewriteRules($rewrite) {
    global $wp_rewrite;

    // add rewrite tokens
    $keytag_token = '%tag%';
    $wp_rewrite->add_rewrite_tag($keytag_token, '(.+)', 'tag=');
```

```

        $keywords_structure = $wp_rewrite->root . "tag/$keytag_token";
        $keywords_rewrite = $wp_rewrite->
generate_rewrite_rules($keywords_structure);

        return ( $rewrite + $keywords_rewrite );
    }

```

Instead of inserting the rewrite rules into the `$rewrite` array itself, Jerome chose to create a second array, `$keywords_rewrite`, using the `WP_Rewrite` function `generate_rewrite_rules()`. Using that function means that the plugin doesn't have to create rewrite rules for extra pages (like `page/2`), or feeds (like `feed/atom`), etc. This array is then appended onto the `$rewrite` array and returned.

A simpler example of this is Ryan Boren's [Feed Director](http://boren.nu/archives/2005/03/29/feed-director-plugin/) (<http://boren.nu/archives/2005/03/29/feed-director-plugin/>) plugin. This simply redirects URLs like <http://example.com/feed.xml> to <http://example.com/feed/rss2>:

```

function feed_dir_rewrite($wp_rewrite) {
    $feed_rules = array(
        'index.rdf' => 'index.php?feed=rdf',
        'index.xml' => 'index.php?feed=rss2',
        '(.+).xml' => 'index.php?feed=' . $wp_rewrite->preg_index(1)
    );

    $wp_rewrite->rules = $feed_rules + $wp_rewrite->rules;
}

```

As the array is so simple here, there is no need to call `generate_rewrite_rules()`. Again, the plugin's rules are added to WordPress'. Notice that as this function filters `generate_rewrite_rules`, it accepts a reference to the entire `$wp_rewrite` object as a parameter, not just the rewrite rules.

Of course, as your adding your rewrite rules to the array before WordPress does anything with them, your plugins rewrite rules will be included in anything WordPress does with the rewrite rules, like write them to the `.htaccess` file.