



The Handbook

Codex

http://codex.wordpress.org/Function_Reference/WP_Query

Version Date

9 August 2005

Function Reference/WP Query

Role of WP_Query

WP_Query is a class defined in `wp-includes/classes.php` that deals with the intricacies of a request to a WordPress blog. The `wp-blog-header.php` (or the WP class in [Version 1.6](#)) gives the `$wp_query` object information defining the current request, and then `$wp_query` determines what type of query it's dealing with (category archive? dated archive? feed? search?), and fetches the requested posts. It retains a lot of information on the request, which can be pulled at a later date.

Methods and Properties

This is the formal documentation of WP_Query. You shouldn't alter the properties directly, but instead use the methods to interact with them. See also the [Interacting with WP_Query](#) for some useful functions that avoid the need to mess around with class internals and global variables.

Properties

`$query`

Holds the query string that was passed to the `$wp_query` object by `wp-blog-header.php` (or the WP class in [Version 1.6](#)). For information on the construction of this query string, see [Wordpress Code Flow](#).

`$query_vars`

An associative array containing the dissected `$query`: an array of the query variables and their respective values.

`$queried_object`

Applicable if the request is a category, author, permalink or Page. Holds information on the requested category, author, post or Page.

`$queried_object_id`

Simply holds the ID of the above property.

`$posts`

Gets filled with the requested posts from the database.

`$post_count`

The number of posts being displayed.

`$current_post`

(available during [The Loop](#)) Index of the post currently being displayed.

`$post`

(available during [The Loop](#)) The post currently being displayed.

`$is_single`, `$is_page`, `$is_archive` etc.

[Booleans](#) dictating what type of request this is. For example, the three just listed represent 'is it a permalink?', 'is it a Page?', 'is it any type of archive page?' respectively.

Methods

(An ampersand (&) before a method name indicates it [returns by reference](#) (<http://www.php.net/manual/en/language.references.return.php>).

`init()`

Initialise the object, set all properties to null, zero or false.

`parse_query($query)`

Takes a query string defining the request, parses it and populates all properties apart from `$posts`, `$post_count`, `$post` and `$current_post`.

`parse_query_vars()`

Reparse the old query string.

`get($query_var)`

Get a named query variable.

`set($query_var, $value)`

Set a named query variable to a specific value.

`&get_posts()`

Fetch and return the requested posts from the database. Also populate `$posts` and `$post_count`.

`next_post()`

(to be used when in [The Loop](#)) Advance onto the next post in `$posts`. Increment `$current_post` and set `$post`.

`the_post()`

(to be used when in [The Loop](#)) Advance onto the next post, and set the global `$post` variable.

`have_posts()`

(to be used when in [The Loop](#), or just before The Loop) Determine if we have posts remaining to be displayed.

`rewind_posts()`

Reset `$current_post` and `$post`.

`&query($query)`

Call `parse_query()` and `get_posts()`. Return the results of `get_posts()`.

`get_queried_object()`

Set `$queried_object` if it's not already set and return it.

`get_queried_object_id()`

Set `$queried_object_id` if it's not already set and return it.

`WP_Query($query = '')` (constructor)

If you provide a query string, call `query()` with it.

Interacting with WP_Query

Most of the time you can find the information you want without actually dealing with the class internals and globals variables. There are a whole bunch of functions that you can call from anywhere that will enable you to get the information you need.

There are two main scenarios you might want to use `WP_Query` in. The first is to find out what type of request WordPress is currently dealing with. The `$is_*` properties are designed to hold this information: use the [Conditional Tags](#) to interact here. This is the more common scenario to plugin writers (the second normally applies to theme writers).

The second is during [The Loop](#). `WP_Query` provides numerous functions for common

tasks within The Loop. To begin with, `have_posts()`, which calls `$wp_query->have_posts()`, is called to see if there are any posts to show. If there are, a `while` loop is begun, using `have_posts()` as the condition. This will iterate around as long as there are posts to show. In each iteration, `the_post()`, which calls `$wp_query->the_post()` is called, setting up internal variables within `$wp_query` and the global `$post` variable (which the [Template Tags](#) rely on), as above. These are the functions you should use when writing a theme file that needs a loop. See also [The Loop](#) and [The Loop in Action](#) for more information