Search WordPress.org

Codex

Codex tools: Log in

Interested in functions, hooks, classes, or methods? Check out the new WordPress Code Reference!

Class Reference/WP Query

Languages: **English** • Italiano • 한국어 • 日本語 中文(简体) • Português do Brasil • (Add your language)

Description

`WP_Query` is a class defined in `wp-includes/class-wp-query.php` that deals with the intricacies of a post's (or page's) request to a WordPress blog. The `wp-blog-header.php` (or the WP class in Version 2.0) gives the `$wp_query` object information defining the current request, and then `$wp_query` determines what type of query it's dealing with (possibly a category archive, dated archive, feed, or search), and fetches the requested posts. It retains a lot of information on the request, which can be pulled at a later date.

Interacting with WP_Query

Most of the time you can find the information you want without actually dealing with the class internals and global 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

Contents

you should use when writing a theme file that needs a loop. See also The Loop and The Loop in Action for more information.

Note: If you use the_post() with your query, you need to run wp_reset_postdata() afterwards to have Template Tags use the main query's current post again.

Note: Ticket #18408 For querying posts in the admin, consider using get_posts() as wp_reset_postdata() might not behave as expected.

Usage

Standard Loop

```
<?php

// The Query
$the_query = new WP_Query( $args );

// The Loop
if ( $the_query->have_posts() ) {
        echo '<ul>';
        while ( $the_query->have_posts() ) {
                $the_query->the_post();
                echo '<li>' . get_the_title() . '</li>';
        }
        echo '</ul>';
        /* Restore original Post Data */
        wp_reset_postdata();
} else {
        // no posts found
}
```

Standard Loop (Alternate)

```
<?php

// the query
$the_query = new WP_Query( $args ); ?>

<?php if ( $the_query->have_posts() ) : ?>

        <!-- pagination here -->

        <!-- the loop -->
        <?php while ( $the_query->have_posts() ) : $the_query->the_post(); ?>
                <h2><?php the_title(); ?></h2>
        <?php endwhile; ?>
        <!-- end of the loop -->

        <!-- pagination here -->

        <?php wp_reset_postdata(); ?>

<?php else : ?>
        <p><?php esc_html_e( 'Sorry, no posts matched your criteria.' ); ?></p>
<?php endif; ?>
```

Multiple Loops

If you have multiple queries, you need to perform multiple loops. Like so...

```
<?php

// The Query
$query1 = new WP_Query( $args );

if ( $query1->have_posts() ) {
	// The Loop
	while ( $query1->have_posts() ) {
		$query1->the_post();
		echo '<li>' . get_the_title() . '</li>';
	}

	/* Restore original Post Data
	 * NB: Because we are using new WP_Query we aren't stomping on the
	 * original $wp_query and it does not need to be reset with
	 * wp_reset_query(). We just need to set the post data back up with
	 * wp_reset_postdata().
	 */
	wp_reset_postdata();
}

/* The 2nd Query (without global var) */
$query2 = new WP_Query( $args2 );

if ( $query2->have_posts() ) {
	// The 2nd Loop
	while ( $query2->have_posts() ) {
		$query2->the_post();
		echo '<li>' . get_the_title( $query2->post->ID ) . '</li>';
	}

	// Restore original Post Data
	wp_reset_postdata();
}

?>
```

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. Also see 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 class.

`$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`

If the request is a category, author, permalink or post / page, holds the corresponding ID.

`$posts`

Gets filled with the requested posts from the database.

`$post_count`

The number of posts being displayed.

`$found_posts`

The total number of posts found matching the current query parameters

`$max_num_pages`

The total number of pages. Is the result of $found_posts / $posts_per_page

`$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`, `$is_archive`, `$is_preview`, `$is_date`, `$is_year`, `$is_month`, `$is_time`, `$is_author`, `$is_category`, `$is_tag`, `$is_tax`, `$is_search`, `$is_feed`, `$is_comment_feed`, `$is_trackback`, `$is_home`, `$is_404`, `$is_comments_popup`, `$is_admin`, `$is_attachment`, `$is_singular`, `$is_robots`, `$is_posts_page`, `$is_paged`

Booleans dictating what type of request this is. For example, the first three represent 'is it a permalink?', 'is it a Page?', 'is it any type of archive page?', respectively. See also Conditional_Tags.

Methods

(An ampersand (&) before a method name indicates it returns by reference.)

`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`. **Note:** This is called during construction if WP_Query is constructed with arguments. It is *not* idempotent and should not be called more than once on the same query object. Doing so may result in a broken query.

`next_post()`

(to be used when in The Loop) Advance onto the next post in `$posts`. Increment `$current_post` and set `$post` to the (new) current post object (note: this does not set the global `$post` variable, only the WP_Query object's instance variable.) Returns the current post object **(This is deprecated pleases use 'next_post_link()')**

`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. Calls `rewind_posts()` and returns false if don't have posts remaining. Because of the rewind, you can't rely on have_posts() staying false. See have_posts() note.

`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.

Parameters

Author Parameters

Show posts associated with certain author.

- **`author`** (*int* | *string*) - use author id or comma-separated list of IDs.
- **`author_name`** (*string*) - use '`user_nicename`' - NOT name.
- **`author__in`** (*array*) - use author id (available since Version 3.7).
- **`author__not_in`** (*array*) - use author id (available since Version 3.7).

Show Posts for one Author

Display posts by author, using author id:

```
$query = new WP_Query( array( 'author' => 123 ) );
```

Display posts by author, using author '`user_nicename`':

```
$query = new WP_Query( array( 'author_name' => 'rami' ) );
```

Show Posts From Several Authors

Display posts from several specific authors:

```
$query = new WP_Query( array( 'author' => '2,6,17,38' ) );
```

Exclude Posts Belonging to an Author

Display all posts *except* those from an author(singular) by prefixing its id with a '-' (minus) sign:

```
$query = new WP_Query( array( 'author' => -12 ) );
```

Multiple Author Handling

Display posts from multiple authors:

```
$query = new WP_Query( array( 'author__in' => array( 2, 6 ) ) );
```

You can also exclude multiple author this way:

```
$query = new WP_Query( array( 'author__not_in' => array( 2, 6 ) ) );
```

Category Parameters

Show posts associated with certain categories.

- **`cat`** (*int*) - use category id.
- **`category_name`** (*string*) - use category slug.
- **`category__and`** (*array*) - use category id.
- **`category__in`** (*array*) - use category id.
- **`category__not_in`** (*array*) - use category id.

Show Posts for One Category

Display posts that have this category (and any children of that category), using category id:

```
$query = new WP_Query( array( 'cat' => 4 ) );
```

Display posts that have this category (and any children of that category), using category slug:

```
$query = new WP_Query( array( 'category_name' => 'staff' ) );
```

Display posts that have this category (not children of that category), using category id:

```
$query = new WP_Query( array( 'category__in' => 4 ) );
```

Show Posts From Several Categories

Display posts that have these categories, using category id:

```
$query = new WP_Query( array( 'cat' => '2,6,17,38' ) );
```

Display posts that have these categories, using category slug:

```
$query = new WP_Query( array( 'category_name' => 'staff,news' ) );
```

Display posts that have "all" of these categories:

```
$query = new WP_Query( array( 'category_name' => 'staff+news' ) );
```

Exclude Posts Belonging to Category

Display all posts *except* those from a category by prefixing its id with a '-' (minus) sign.

```
$query = new WP_Query( array( 'cat' => '-12,-34,-56' ) );
```

Multiple Category Handling

Display posts that are in multiple categories. This shows posts that are in both categories 2 and 6:

```
$query = new WP_Query( array( 'category__and' => array( 2, 6 ) ) );
```

To display posts from either category 2 OR 6, you could use `cat` as mentioned above, or by using `category__in` (note this does not show posts from any children of these categories):

```
$query = new WP_Query( array( 'category__in' => array( 2, 6 ) ) );
```

You can also exclude multiple categories this way:

```
$query = new WP_Query( array( 'category__not_in' => array( 2, 6 ) ) );
```

Tag Parameters

Show posts associated with certain tags.

- **`tag`** (*string*) - use tag slug.
- **`tag_id`** (*int*) - use tag id.
- **`tag__and`** (*array*) - use tag ids.
- **`tag__in`** (*array*) - use tag ids.
- **`tag__not_in`** (*array*) - use tag ids.
- **`tag_slug__and`** (*array*) - use tag slugs.
- **`tag_slug__in`** (*array*) - use tag slugs.

Show Posts for One Tag

Display posts that have this tag, using tag slug:

```
$query = new WP_Query( array( 'tag' => 'cooking' ) );
```

Display posts that have this tag, using tag id:

```
$query = new WP_Query( array( 'tag_id' => 13 ) );
```

Show Posts From Several Tags

Display posts that have "either" of these tags:

```
$query = new WP_Query( array( 'tag' => 'bread,baking' ) );
```

Display posts that have "all" of these tags:

```
$query = new WP_Query( array( 'tag' => 'bread+baking+recipe' ) );
```

Multiple Tag Handling

Display posts that are tagged with both tag id 37 and tag id 47:

```
$query = new WP_Query( array( 'tag__and' => array( 37, 47 ) ) );
```

To display posts from either tag id 37 or 47, you could use `tag` as mentioned above, or explicitly specify by using `tag__in`:

```
$query = new WP_Query( array( 'tag__in' => array( 37, 47 ) ) );
```

Display posts that do not have any of the two tag ids 37 and 47:

```
$query = new WP_Query( array( 'tag__not_in' => array( 37, 47 ) ) );
```

The `tag_slug__in` and `tag_slug__and` behave much the same, except match against the tag's slug.

Taxonomy Parameters

Show posts associated with certain taxonomy.

- **`{tax}`** (*string*) - use taxonomy slug. (**Deprecated** since Version 3.1 in favor of '`tax_query`').
- **`tax_query`** (*array*) - use taxonomy parameters (available since Version 3.1).
  - **`relation`** (*string*) - The logical relationship between each inner taxonomy array when there is more than one. Possible values are 'AND', 'OR'. Do not use with a single inner taxonomy array. Default value is 'AND'.
    - **`taxonomy`** (*string*) - Taxonomy.
    - **`field`** (*string*) - Select taxonomy term by. Possible values are 'term_id', 'name', 'slug' or 'term_taxonomy_id'. Default value is 'term_id'.
    - **`terms`** (*int/string/array*) - Taxonomy term(s).
    - **`include_children`** (*boolean*) - Whether or not to include children for hierarchical taxonomies. Defaults to true.
    - **`operator`** (*string*) - Operator to test. Possible values are 'IN', 'NOT IN', 'AND', 'EXISTS' and 'NOT EXISTS'. Default value is 'IN'.

Note: If `WP_Query` determines that the result will be singular (`is_singular()` is `true`), it will ignore the `tax_query` parameter. To modify `tax_query`, use the `posts_clauses` filter to add the required SQL statements.

Important Note: `tax_query` takes an **array** of tax query arguments **arrays** (it takes an array of arrays). This construct allows you to query multiple taxonomies by using the **`relation`** parameter in the first (outer) array to describe the boolean relationship between the taxonomy arrays.

Simple Taxonomy Query:

Display **posts** tagged with **bob**, under **people** custom taxonomy:

```
$args = array(
	'post_type' => 'post',
	'tax_query' => array(
		array(
			'taxonomy' => 'people',
			'field'    => 'slug',
			'terms'    => 'bob',
		),
	),
);
$query = new WP_Query( $args );
```

Multiple Taxonomy Handling:

Display **posts** from several custom taxonomies:

```
$args = array(
	'post_type' => 'post',
	'tax_query' => array(
		'relation' => 'AND',
		array(
			'taxonomy' => 'movie_genre',
			'field'    => 'slug',
			'terms'    => array( 'action', 'comedy' ),
		),
		array(
			'taxonomy' => 'actor',
			'field'    => 'term_id',
			'terms'    => array( 103, 115, 206 ),
			'operator' => 'NOT IN',
		),
	),
);
$query = new WP_Query( $args );
```

Display **posts** that are in the **quotes** category OR have the **quote** format:

```
$args = array(
	'post_type' => 'post',
	'tax_query' => array(
		'relation' => 'OR',
		array(
			'taxonomy' => 'category',
			'field'    => 'slug',
			'terms'    => array( 'quotes' ),
		),
		array(
			'taxonomy' => 'post_format',
			'field'    => 'slug',
			'terms'    => array( 'post-format-quote' ),
		),
	),
);
$query = new WP_Query( $args );
```

Nested Taxonomy Handling:

The `'tax_query'` clauses can be nested, to create more complex queries. Example: Display **posts** that are in the **quotes** category OR both have the **quote** post format AND are in the **wisdom** category:

```
$args = array(
	'post_type' => 'post',
	'tax_query' => array(
		'relation' => 'OR',
		array(
			'taxonomy' => 'category',
			'field'    => 'slug',
			'terms'    => array( 'quotes' ),
		),
		array(
			'relation' => 'AND',
                                array(
                                        'taxonomy' => 'post_format',
                                        'field'    => 'slug',
                                        'terms'    => array( 'post-format-quote' ),
                                ),
                                array(
                                        'taxonomy' => 'category',
                                        'field'    => 'slug',
                                        'terms'    => array( 'wisdom' ),
                                ),
                        ),
                ),
);
$query = new WP_Query( $args );
```

Search Parameter

Show posts based on a keyword search.

- **`s`** (*string*) - Search keyword.

Show Posts based on a keyword search

Display posts that match the search term "keyword":

```
$query = new WP_Query( array( 's' => 'keyword' ) );
```

Prepending a term with a hyphen will exclude posts matching that term. Eg, 'pillow -sofa' will return posts containing 'pillow' but not 'sofa' (available since Version 4.4).

Post & Page Parameters

Display content based on post and page parameters. Remember that default `post_type` is only set to display posts but not pages.

- **`p`** (*int*) - use post id. Default post type is post.
- **`name`** (*string*) - use post slug.
- **`title`** (*string*) - use post title (available with Version 4.4).
- **`page_id`** (*int*) - use page id.
- **`pagename`** (*string*) - use page slug.
- **`post_parent`** (*int*) - use page id to return only child pages. Set to 0 to return only top-level entries.
- **`post_parent__in`** (*array*) - use post ids. Specify posts whose parent is in an array. (available since Version 3.6)
- **`post_parent__not_in`** (*array*) - use post ids. Specify posts whose parent is not in an array. Like post__in/post__not_in, id's that are present in post_parent__in will over-ride ids specified in post_parent__not_in (available since Version 3.6)
- **`post__in`** (*array*) - use post ids. Specify posts to retrieve. **ATTENTION** If you use sticky posts, they will be included (prepended!) in the posts you retrieve whether you want it or not. To suppress this behaviour use ignore_sticky_posts.
- **`post__not_in`** (*array*) - use post ids. Specify post NOT to retrieve. If this is used in the same query as post__in, it will be ignored.
- **`post_name__in`** (*array*) - use post slugs. Specify posts to retrieve. (available since Version 4.4)

NOTE: Ticket #28099 Passing an empty array to `post__in` will return `have_posts()` as true (and all posts will be returned). Logic should be used before hand to determine if `WP_Query` should be used in the event that the array being passed to `post__in` is empty.

Show Post/Page by ID

Display post by ID:

```
$query = new WP_Query( array( 'p' => 7 ) );
//for page, use:              'page_id'=>7
```

NOTE: Beware of ID = 0. This might return results.

Show Post/Page by Slug

Display post by `slug`:

```
$query = new WP_Query( array( 'name' => 'about-my-life' ) );
//for page, use:              'pagename'=>'about-my-life'
```

Show Child Posts/Pages

Display child page using the slug of the parent and the child page, separated by a slash (e.g. 'parent_slug/child_slug'):

```
$query = new WP_Query( array( 'pagename' => 'contact_us/canada' ) );
```

Display child pages using parent page ID:

```
$query = new WP_Query( array( 'post_parent' => 93 ) );
```

Display only top-level pages, exclude all child pages:

```
$query = new WP_Query( array( 'post_parent' => 0 ) );
```

Display posts whose parent is in an array:

```
$query = new WP_Query( array( 'post_parent__in' => array( 2, 5, 12, 14, 20 ) ) );
```

Multiple Posts/Pages Handling

Display only the specific posts:

```
$query = new WP_Query( array( 'post_type' => 'page', 'post__in' => array( 2, 5, 12, 14, 20 ) ) );
```

Display all posts but NOT the specified ones:

```
$query = new WP_Query( array( 'post_type' => 'post', 'post__not_in' => array( 2, 5, 12, 14, 20 ) ) );
```

Note: you cannot combine `post__in` and `post__not_in` in the same query.

Also note, not to pass comma separated string

```
'post__not_in' => array( '1,2,3' )  // <--- this wont work
```

Password Parameters

Show content based on post and page parameters. Remember that default `post_type` is only set to display posts but not pages.

- **`has_password`** (*bool*) - true for posts with passwords ; false for posts without passwords ; null for all posts with and without passwords (available since Version 3.9).
- **`post_password`** (*string*) - show posts with a particular password (available since Version 3.9)

Show Posts with/without passwords

Display only password protected posts:

```
$query = new WP_Query( array( 'has_password' => true ) );
```

Display only posts without passwords:

```
$query = new WP_Query( array( 'has_password' => false ) );
```

Display only posts with and without passwords:

```
$query = new WP_Query( array( 'has_password' => null ) );
```

Show Posts with particular password

Display posts with 'zxcvbn' password:

```
$query = new WP_Query( array( 'post_password' => 'zxcvbn' ) );
```

Type Parameters

Show posts associated with certain type.

- **`post_type`** (*string* / *array*) - use post types. Retrieves posts by Post Types, default value is '`post`'. If '`tax_query`' is set for a query, the default value becomes '`any`';
  - '`post`' - a post.

- 'page' - a page.
- 'revision' - a revision.
- 'attachment' - an attachment. Whilst the default WP_Query post_status is 'publish', attachments have a default post_status of 'inherit'. This means no attachments will be returned unless you also explicitly set post_status to 'inherit' or 'any'. (See post_status, below)
- 'nav_menu_item' - a navigation menu item
- 'any' - retrieves any type except revisions and types with 'exclude_from_search' set to true.
- Custom Post Types (e.g. movies)

Show Post by Type

Display only pages:

```
$query = new WP_Query( array( 'post_type' => 'page' ) );
```

Display 'any' post type (retrieves any type except revisions and types with 'exclude_from_search' set to TRUE):

```
$query = new WP_Query( array( 'post_type' => 'any' ) );
```

Display multiple post types, including custom post types:

```
$args = array(
        'post_type' => array( 'post', 'page', 'movie', 'book' )
);
$query = new WP_Query( $args );
```

Status Parameters

Show posts associated with certain status.

- **post_status** (*string* / *array*) - use post status. Retrieves posts by Post Status. Default value is 'publish', but if the user is logged in, 'private' is added. Public custom statuses are also included by default. And if the query is run in an admin context (administration area or AJAX call), protected statuses are added too. By default protected statuses are 'future', 'draft' and 'pending'.
  - 'publish' - a published post or page.
  - 'pending' - post is pending review.
  - 'draft' - a post in draft status.
  - 'auto-draft' - a newly created post, with no content.
  - 'future' - a post to publish in the future.
  - 'private' - not visible to users who are not logged in.
  - 'inherit' - a revision. see get_children.
  - 'trash' - post is in trashbin (available since Version 2.9).
  - 'any' - retrieves any status except those from post statuses with 'exclude_from_search' set to true (i.e. trash and auto-draft).

Show Post by Status

Display only drafts:

```
$query = new WP_Query( array( 'post_status' => 'draft' ) );
```

Display multiple post status:

```
$args = array(
        'post_status' => array( 'pending', 'draft', 'future' )
);
$query = new WP_Query( $args );
```

Display all attachments:

```
$args = array(
        'post_status' => 'any',
        'post_type'   => 'attachment'
);
$query = new WP_Query( $args );
```

Comment Parameters

@since Version 4.9 Introduced the `$comment_count` parameter.

- **`comment_count`** (*int*) - The amount of comments your CPT has to have (Search operator will do a '=' operation)
- **`comment_count`** (*array*) -
  - **`value`** (*int*) - The amount of comments your CPT has to have when comparing
  - **`compare`** (*string*) - The search operator. Possible values are '=', '!=', '>', '>=', '<', '<='. Default value is '='.

Simple Comment count Query:

Display **posts** with with **20** comments:

```
$args = array(
        'post_type' => 'post',
        'comment_count' => 20,
        )
);
$query = new WP_Query( $args );
```

Advanced Comment count Query:

Display **posts** with at least **25** comments:

```
$args = array(
        'post_type' => 'post',
        'comment_count' => array(
                array(
                        'value' => 25,
                        'compare' => '>=',
                ),
        )
);
$query = new WP_Query( $args );
```

Pagination Parameters

- **`nopaging`** (*boolean*) - show all posts or use pagination. Default value is 'false', use paging.
- **`posts_per_page`** (*int*) - number of post to show per page (available since Version 2.1, replaced **`showposts`** parameter). Use `'posts_per_page'=>-1` to show all posts (the `'offset'` parameter is ignored with a `-1` value). Set the 'paged' parameter if pagination is off after using this parameter. *Note*: if the query is in a feed, wordpress overwrites this parameter with the stored 'posts_per_rss' option. To reimpose the limit, try using the 'post_limits' filter, or filter 'pre_option_posts_per_rss' and return -1
- **`posts_per_archive_page`** (*int*) - number of posts to show per page - on archive pages only. Over-rides **`posts_per_page`** and **`showposts`** on pages where `is_archive()` or `is_search()` would be true.
- **`offset`** (*int*) - number of post to *displace* or pass over. *Warning*: Setting the offset parameter overrides/ignores the paged parameter and breaks pagination (Click here for a workaround). The `'offset'` parameter is ignored when `'posts_per_page'=>-1` (show all posts) is used.
- **`paged`** (*int*) - number of page. Show the posts that would normally show up just on page X when using the "Older Entries" link.
- **`page`** (*int*) - number of page for a static front page. Show the posts that would normally show up just on page X of a Static Front Page.
- **`ignore_sticky_posts`** (*boolean*) - ignore post stickiness (available since Version 3.1, replaced **`caller_get_posts`** parameter). `false` (default): move sticky posts to the start of the set. `true`: do not move sticky posts to the start of the set.

Show x Posts per page

```
$query = new WP_Query( array( 'posts_per_page' => 3 ) );     // Display 3 posts per page
```

```
$query = new WP_Query( array( 'posts_per_page' => -1 ) );    // Display all posts in one page
```

Display all posts by disabling pagination:

```
$query = new WP_Query( array( 'nopaging' => true ) );
```

Pass over Posts

Display posts from the 4th one:

```
$query = new WP_Query( array( 'offset' => 3 ) );
```

Display 5 posts per page which follow the 3 most recent posts:

```
$query = new WP_Query( array( 'posts_per_page' => 5, 'offset' => 3 ) );
```

Show Posts from page x

```
$query = new WP_Query( array( 'paged' => 6 ) );   // page number 6
```

Show Posts from Current Page

```
$query = new WP_Query( array( 'paged' => get_query_var( 'paged' ) ) );
```

Display posts from the current page and set the 'paged' parameter to 1 when the query variable is not set (first page).

```
$paged = ( get_query_var('paged') ) ? get_query_var('paged') : 1;
$query = new WP_Query( array( 'paged' => $paged ) );
```

Pagination Note: Use `get_query_var('page');` if you want your query to work in a Page template that you've set as your static front page. The query variable 'page' also holds the pagenumber for a single paginated Post or Page that includes the `<!--nextpage-->` Quicktag in the post content.

Display posts from current page on a static front page:

```
$paged = ( get_query_var('page') ) ? get_query_var('page') : 1;
$query = new WP_Query( array( 'paged' => $paged ) );
```

Show Sticky Posts

Display just the first sticky post:

```
$sticky = get_option( 'sticky_posts' );
$query = new WP_Query( array( 'p' => $sticky[0] ) );
```

Display just the first sticky post, if none return the last post published:

```
$sticky = get_option( 'sticky_posts' );
$args = array(
        'posts_per_page'      => 1,
        'post__in'            => $sticky,
        'ignore_sticky_posts' => 1,
);
$query = new WP_Query( $args );
```

Display just the first sticky post, add this line to above block:

```
if ( $sticky[0] ) {
        // insert here your stuff...
}
```

Don't Show Sticky Posts

Exclude all sticky posts from the query:

```
$query = new WP_Query( array( 'post__not_in' => get_option( 'sticky_posts' ) ) );
```

Return ALL posts within the category, but don't show ("ignore") sticky posts at the top (They will still show in their natural position, e.g. by date):

```
$query = new WP_Query( array( 'ignore_sticky_posts' => 1, 'posts_per_page' => 3, 'cat' => 6 );
```

Exclude sticky posts from a category. Return posts within the category, but exclude sticky posts completely, and adhere to paging rules:

```
$paged = get_query_var( 'paged' ) ? get_query_var( 'paged' ) : 1;
$sticky = get_option( 'sticky_posts' );
$args = array(
        'cat'                 => 3,
        'ignore_sticky_posts' => 1,
        'post__not_in'        => $sticky,
        'paged'               => $paged,
);
$query = new WP_Query( $args );
```

Order & Orderby Parameters

Sort retrieved posts.

- **`order`** (*string* | *array*) - Designates the ascending or descending order of the '`orderby`' parameter. Defaults to 'DESC'. An array can be used for multiple order/orderby sets.
  - '`ASC`' - ascending order from lowest to highest values (1, 2, 3; a, b, c).
  - '`DESC`' - descending order from highest to lowest values (3, 2, 1; c, b, a).

- **`orderby`** (*string* | *array*) - Sort retrieved posts by parameter. Defaults to 'date (post_date)'. One or more options can be passed.
  - '`none`' - No order (available since Version 2.8).
  - '`ID`' - Order by post id. Note the capitalization.
  - '`author`' - Order by author. ('`post_author`' is also accepted.)
  - '`title`' - Order by title. ('`post_title`' is also accepted.)
  - '`name`' - Order by post name (post slug). ('`post_name`' is also accepted.)
  - '`type`' - Order by post type (available since Version 4.0). ('`post_type`' is also accepted.)
  - '`date`' - Order by date. ('`post_date`' is also accepted.)

- 'modified' - Order by last modified date. ('post_modified' is also accepted.)
- 'parent' - Order by post/page parent id. ('post_parent' is also accepted.)
- 'rand' - Random order. You can also use 'RAND(x)' where 'x' is an integer seed value. Note an "order" parameter needs to be present for "orderby" rand to work.
- 'comment_count' - Order by number of comments (available since Version 2.9).
- 'relevance' - Order by search terms in the following order: First, whether the entire sentence is matched. Second, if all the search terms are within the titles. Third, if any of the search terms appear in the titles. And, fourth, if the full sentence appears in the contents.
- 'menu_order' - Order by Page Order. Used most often for Pages (*Order* field in the Edit Page Attributes box) and for Attachments (the integer fields in the Insert / Upload Media Gallery dialog), but could be used for any post type with distinct 'menu_order' values (they all default to 0).
- 'meta_value' - Note that a 'meta_key=keyname' must also be present in the query. Note also that the sorting will be alphabetical which is fine for strings (i.e. words), but can be unexpected for numbers (e.g. 1, 3, 34, 4, 56, 6, etc, rather than 1, 3, 4, 6, 34, 56 as you might naturally expect). Use 'meta_value_num' instead for numeric values. You may also specify 'meta_type' if you want to cast the meta value as a specific type. Possible values are 'NUMERIC', 'BINARY', 'CHAR', 'DATE', 'DATETIME', 'DECIMAL', 'SIGNED', 'TIME', 'UNSIGNED', same as in '$meta_query'. When using 'meta_type' you can also use 'meta_value_*' accordingly. For example, when using DATETIME as 'meta_type' you can use 'meta_value_datetime' to define order structure.
- 'meta_value_num' - Order by numeric meta value (available since Version 2.8). Also note that a 'meta_key=keyname' must also be present in the query. This value allows for numerical sorting as noted above in 'meta_value'.
- 'post__in' - Preserve post ID order given in the 'post__in' array (available since Version 3.5). **Note** - the value of the order parameter does not change the resulting sort order.
- 'post_name__in' - Preserve post slug order given in the 'post_name__in' array (available since Version 4.6). **Note** - the value of the order parameter does not change the resulting sort order.
- 'post_parent__in' - Preserve post parent order given in the 'post_parent__in' array (available since Version 4.6). **Note** - the value of the order parameter does not change the resulting sort order.

Show Posts sorted by Title, Descending order

Display posts sorted by post 'title' in a descending order:

```
$args = array(
        'orderby' => 'title',
        'order'   => 'DESC',
);
$query = new WP_Query( $args );
```

Display posts sorted by 'menu_order' with a fallback to post 'title', in a descending order:

```
$args = array(
        'orderby' => 'menu_order title',
        'order'   => 'DESC',
);
$query = new WP_Query( $args );
```

Show Random Post

Display one random post:

```
$args = array(
        'orderby'        => 'rand',
        'posts_per_page' => 1,

);
$query = new WP_Query( $args );
```

Show Popular Posts

Display posts ordered by comment count:

```
$args = array(
        'orderby' => 'comment_count'
);
$query = new WP_Query( $args );
```

Show Products sorted by Price

Display posts with 'Product' type ordered by 'Price' custom field:

```
$args = array(
        'post_type' => 'product',
        'orderby'   => 'meta_value_num',
        'meta_key'  => 'price',
);
$query = new WP_Query( $args );
```

Multiple 'orderby' values

Display pages ordered by 'title' and 'menu_order'. (title is dominant):

```
$args = array(
        'post_type' => 'page',
        'orderby'   => 'title menu_order',
        'order'     => 'ASC',
);
$query = new WP_Query( $args );
```

Multiple 'orderby' values using an array

Display pages ordered by 'title' and 'menu_order' with different sort orders (ASC/DESC) (available since Version 4.0):

```
$args = array(
        'orderby' => array( 'title' => 'DESC', 'menu_order' => 'ASC' )
);
$query = new WP_Query( $args );
```

- A more powerful ORDER BY in WordPress 4.0

Mulitiple orderby/order pairs

```
$args = array(
	'orderby'  => array( 'meta_value_num' => 'DESC', 'title' => 'ASC' ),
	'meta_key' => 'age'
);
$query = new WP_Query( $args );
```

'orderby' with 'meta_value' and custom post type

Display posts of type 'my_custom_post_type', ordered by 'age', and filtered to show only ages 3 and 4 (using meta_query).

```
$args = array(
	'post_type'  => 'my_custom_post_type',
	'meta_key'   => 'age',
	'orderby'    => 'meta_value_num',
	'order'      => 'ASC',
	'meta_query' => array(
		array(
			'key'     => 'age',
			'value'   => array( 3, 4 ),
			'compare' => 'IN',
		),
	),
);
$query = new WP_Query( $args );
```

'orderby' with multiple 'meta_key's

If you wish to order by two different pieces of postmeta (for example, City first and State second), you need to combine and link your meta query to your orderby array using 'named meta queries'. See the example below:

```
$q = new WP_Query( array(
    'meta_query' => array(
        'relation' => 'AND',
        'state_clause' => array(
            'key' => 'state',
            'value' => 'Wisconsin',
        ),
        'city_clause' => array(
            'key' => 'city',
            'compare' => 'EXISTS',
        ),
    ),
    'orderby' => array(
        'city_clause' => 'ASC',
        'state_clause' => 'DESC',
    ),
) );
```

Props cybmeta on WPSE for this example.

Date Parameters

Show posts associated with a certain time and date period.

- **year** (*int*) - 4 digit year (e.g. 2011).
- **monthnum** (*int*) - Month number (from 1 to 12).
- **w** (*int*) - Week of the year (from 0 to 53). Uses MySQL WEEK command. The mode is dependent on the "start_of_week" option.
- **day** (*int*) - Day of the month (from 1 to 31).
- **hour** (*int*) - Hour (from 0 to 23).
- **minute** (*int*) - Minute (from 0 to 60).
- **second** (*int*) - Second (0 to 60).
- **m** (*int*) - YearMonth (For e.g.: **201307**).

- **date_query** (*array*) - Date parameters (available since Version 3.7).
  - **year** (*int*) - 4 digit year (e.g. 2011).
  - **month** (*int*) - Month number (from 1 to 12).
  - **week** (*int*) - Week of the year (from 0 to 53).
  - **day** (*int*) - Day of the month (from 1 to 31).
  - **hour** (*int*) - Hour (from 0 to 23).
  - **minute** (*int*) - Minute (from 0 to 59).
  - **second** (*int*) - Second (0 to 59).
  - **after** (*string/array*) - Date to retrieve posts after. Accepts `strtotime()`-compatible string, or array of 'year', 'month', 'day' values:
    - **year** (*string*) Accepts any four-digit year. Default is empty.
    - **month** (*string*) The month of the year. Accepts numbers 1-12. Default: 12.
    - **day** (*string*) The day of the month. Accepts numbers 1-31. Default: last day of month.

  - **before** (*string/array*) - Date to retrieve posts before. Accepts `strtotime()`-compatible string, or array of 'year', 'month', 'day' values:
    - **year** (*string*) Accepts any four-digit year. Default is empty.
    - **month** (*string*) The month of the year. Accepts numbers 1-12. Default: 1.
    - **day** (*string*) The day of the month. Accepts numbers 1-31. Default: 1.

  - **inclusive** (*boolean*) - For after/before, whether exact value should be matched or not'.
  - **compare** (*string*) - See WP_Date_Query::get_compare().
  - **column** (*string*) - Column to query against. Default: 'post_date'.
  - **relation** (*string*) - OR or AND, how the sub-arrays should be compared. Default: AND.

Returns posts dated December 12, 2012:

```
$query = new WP_Query( 'year=2012&monthnum=12&day=12' );
```

or:

```
$args = array(
	'date_query' => array(
		array(
			'year'  => 2012,
			'month' => 12,
			'day'   => 12,
                ),
        ),
);
$query = new WP_Query( $args );
```

Returns posts for today:

```
$today = getdate();
$query = new WP_Query( 'year=' . $today['year'] . '&monthnum=' . $today['mon'] . '&day=' .
$today['mday'] );
```

or:

```
$today = getdate();
$args = array(
        'date_query' => array(
                array(
                        'year'  => $today['year'],
                        'month' => $today['mon'],
                        'day'   => $today['mday'],
                ),
        ),
);
$query = new WP_Query( $args );
```

Returns posts for this week:

```
$week = date( 'W' );
$year = date( 'Y' );
$query = new WP_Query( 'year=' . $year . '&w=' . $week );
```

or:

```
$args = array(
        'date_query' => array(
                array(
                        'year' => date( 'Y' ),
                        'week' => date( 'W' ),
                ),
        ),
);
$query = new WP_Query( $args );
```

Return posts between 9AM to 5PM on weekdays

```
$args = array(
        'date_query' => array(
                array(
                        'hour'      => 9,
                        'compare'   => '>=',
                ),
                array(
                        'hour'      => 17,
                        'compare'   => '<=',
                ),
                array(
                        'dayofweek' => array( 2, 6 ),
                        'compare'   => 'BETWEEN',
                ),
        ),
        'posts_per_page' => -1,
);
$query = new WP_Query( $args );
```

Return posts from January 1st to February 28th

```
$args = array(
        'date_query' => array(
                array(
                        'after'     => 'January 1st, 2013',
                        'before'    => array(
                                'year'  => 2013,
                                'month' => 2,
                                'day'   => 28,
                        ),
                        'inclusive' => true,
                ),
        ),
        'posts_per_page' => -1,
);
$query = new WP_Query( $args );
```

Note that if a `strtotime()`-compatible string with just a date was passed in the `before` parameter, this will be converted to 00:00:00 on that date. In this case, even if `inclusive` was set to true, the date would not be included in the query. If you want a before date to be inclusive, include the time as well, such as `'before' => '2013-02-28 23:59:59'`, or use the array format, which is adjusted automatically if `inclusive` is set.

Return posts made over a year ago but modified in the past month

```
$args = array(
        'date_query' => array(
                array(
                        'column' => 'post_date_gmt',
                        'before' => '1 year ago',
                ),
                array(
                        'column' => 'post_modified_gmt',
                        'after'  => '1 month ago',
                ),
        ),
        'posts_per_page' => -1,
);
$query = new WP_Query( $args );
```

The `'date_query'` clauses can be nested, in order to construct complex queries. See #Taxonomy Parameters for details on the syntax.

Custom Field Parameters

Show posts associated with a certain custom field.

This part of the query is parsed by `WP_Meta_Query`, so check the docs for it as well in case this list of arguments isn't up to date.

- **`meta_key`** (*string*) - Custom field key.
- **`meta_value`** (*string*) - Custom field value.
- **`meta_value_num`** (*number*) - Custom field value.
- **`meta_compare`** (*string*) - Operator to test the '`meta_value`'. Possible values are '=', '!=', '>', '>=', '<', '<=', 'LIKE', 'NOT LIKE', 'IN', 'NOT IN', 'BETWEEN', 'NOT BETWEEN', 'NOT EXISTS', 'REGEXP', 'NOT REGEXP' or 'RLIKE'. Default value is '='.

- **`meta_query`** (*array*) - Custom field parameters (available since Version 3.1).
  - **`relation`** (*string*) - The logical relationship between each inner meta_query array when there is more than one. Possible values are 'AND', 'OR'. Do not use with a single inner meta_query array.

`meta_query` also contains one or more arrays with the following keys:

- **`key`** (*string*) - Custom field key.
- **`value`** (*string|array*) - Custom field value. It can be an array only when **`compare`** is `'IN'`, `'NOT IN'`, `'BETWEEN'`, or `'NOT BETWEEN'`. You don't have to specify a value when using the `'EXISTS'` or `'NOT EXISTS'` comparisons in WordPress 3.9 and up. (**Note:** Due to bug #23268, `value` is required for `NOT EXISTS` comparisons to work correctly **prior to 3.9**. You must supply *some* string for the `value` parameter. An empty string or NULL will NOT work. However, any other string will do the trick and will NOT show up in your SQL when using `NOT EXISTS`. Need inspiration? How about `'bug #23268'`.)
- **`compare`** (*string*) - Operator to test. Possible values are '=', '!=', '>', '>=', '<', '<=', 'LIKE', 'NOT LIKE', 'IN', 'NOT IN', 'BETWEEN', 'NOT BETWEEN', 'EXISTS' and 'NOT EXISTS'. Default value is '='.
- **`type`** (*string*) - Custom field type. Possible values are 'NUMERIC', 'BINARY', 'CHAR', 'DATE', 'DATETIME', 'DECIMAL', 'SIGNED', 'TIME', 'UNSIGNED'. Default value is 'CHAR'. You can also specify precision and scale for the 'DECIMAL' and 'NUMERIC' types (for example, 'DECIMAL(10,5)' or 'NUMERIC(10)' are valid).

The 'type' DATE works with the 'compare' value BETWEEN only if the date is stored at the format YYYY-MM-DD and tested with this format.

Important Note: `meta_query` takes an **array** of meta query arguments **arrays** (it takes an array of arrays) - you can see this in the examples below. This construct allows you to query multiple metadatas by using the **`relation`** parameter in the first (outer) array to describe the boolean relationship between the meta queries. Accepted arguments are 'AND', 'OR'. The default is 'AND'.

Simple Custom Field Query:

Display posts where the custom field key is 'color', regardless of the custom field value:

```
$query = new WP_Query( array( 'meta_key' => 'color' ) );
```

Display posts where the custom field value is 'blue', regardless of the custom field key:

```
$query = new WP_Query( array( 'meta_value' => 'blue' ) );
```

Display Page where the custom field value is 'blue', regardless of the custom field key:

```
$args = array(
	'meta_value' => 'blue',
	'post_type'  => 'page'
);
$query = new WP_Query( $args );
```

Display posts where the custom field key is 'color' and the custom field value is 'blue':

```
$args = array(
	'meta_key'   => 'color',
	'meta_value' => 'blue'
);
$query = new WP_Query( $args );
```

Display posts where the custom field key is 'color' and the custom field value IS NOT 'blue':

```
$args = array(
	'meta_key'     => 'color',
	'meta_value'   => 'blue',
	'meta_compare' => '!='
);
$query = new WP_Query( $args );
```

Display posts where the custom field value is a number. Displays only posts where that number is less than 10. (`WP_Query` uses this equation to compare: `$post_meta . $args['meta_compare'] . $args['meta_value']`, where `'$post_meta'` is the value of the custom post meta stored in each post; the actual equation with the values filled in: `$post_meta < 10`)

```
$args = array(
	'post_type' => 'post',
	'meta_key' => 'number',
	'meta_value_num' => 10,
	'meta_compare' => '<',
);
$query = new WP_Query( $args );
```

Display posts where the custom field key is a set date and the custom field value is now. Displays only posts which date has not passed.

```
$args = array(
	'post_type'    => 'event',
	'meta_key'     => 'event_date',
	'meta_value'   => date( "Ymd" ), // change to how "event date" is stored
	'meta_compare' => '>',
);
$query = new WP_Query( $args );
```

Display 'product'(s) where the custom field key is 'price' and the custom field value that is LESS THAN OR EQUAL TO 22.
By using the 'meta_value' parameter the value 99 will be considered greater than 100 as the data are stored as 'strings', not 'numbers'. For number comparison use 'meta_value_num'.

```
$args = array(
        'meta_key'     => 'price',
        'meta_value'   => '22',
        'meta_compare' => '<=',
        'post_type'    => 'product'
);
$query = new WP_Query( $args );
```

Display posts with a custom field value of zero (0), regardless of the custom field key:

```
$args = array(
        'meta_value' => '_wp_zero_value'
);
$query = new WP_Query( $args );
```

Single Custom Field Handling:

Display posts from a single custom field:

```
$args = array(
        'post_type'  => 'product',
        'meta_query' => array(
                array(
                        'key'     => 'color',
                        'value'   => 'blue',
                        'compare' => 'NOT LIKE',
                ),
        ),
);
$query = new WP_Query( $args );
```

(Note that meta_query expects nested arrays, even if you only have one query.)

Multiple Custom Field Handling:

Display posts that have meta key 'color' NOT LIKE value 'blue' OR meta key 'price' with values BETWEEN 20 and 100:

```
$args = array(
        'post_type'  => 'product',
        'meta_query' => array(
                'relation' => 'OR',
                array(
                        'key'     => 'color',
                        'value'   => 'blue',
                        'compare' => 'NOT LIKE',
                ),
                array(
                        'key'     => 'price',
                        'value'   => array( 20, 100 ),
                        'type'    => 'numeric',
                        'compare' => 'BETWEEN',
                ),
        ),
);
$query = new WP_Query( $args );
```

The `'meta_query'` clauses can be nested in order to construct complex queries. For example, show products where **color=orange** OR **color=red&size=small** translates to the following:

```
$args = array(
	'post_type'  => 'product',
	'meta_query' => array(
		'relation' => 'OR',
		array(
			'key'     => 'color',
			'value'   => 'orange',
			'compare' => '=',
		),
		array(
			'relation' => 'AND',
			array(
				'key' => 'color',
				'value' => 'red',
				'compare' => '=',
			),
			array(
				'key' => 'size',
				'value' => 'small',
				'compare' => '=',
			),
		),
	),
);
$query = new WP_Query( $args );
```

Permission Parameters

- **`perm`** (*string*) - User permission.

Show posts if user has the appropriate capability:

Display published and private posts, if the user has the appropriate capability:

```
$args = array(
	'post_status' => array( 'publish', 'private' ),
	'perm'        => 'readable',
);
$query = new WP_Query( $args );
```

Mime Type Parameters

Used with the attachments post type.

- **`post_mime_type`** (*string/array*) - Allowed mime types.

Get attachments that are *gif* images:

Get gif images and remember that by default the attachment's post_status is set to **inherit**.

```
$args = array(
	'post_type'      => 'attachment',
	'post_status'    => 'inherit',
        'post_mime_type' => 'image/gif',
);
$query = new WP_Query( $args );
```

Get attachments that are not images:

To exclude certain mime types you first need to get all mime types using get_allowed_mime_types() and run a difference between arrays of what you want and the allowed mime types with array_diff().

```
$unsupported_mimes  = array( 'image/jpeg', 'image/gif', 'image/png', 'image/bmp', 'image/tiff',
'image/x-icon' );
$all_mimes          = get_allowed_mime_types();
$accepted_mimes     = array_diff( $all_mimes, $unsupported_mimes );
$args               = array(
    'post_type'         => 'attachment',
    'post_status'       => 'inherit',
    'post_mime_type'    => $accepted_mimes,
);
$query              = new WP_Query( $query_args );
```

Caching Parameters

Stop the data retrieved from being added to the cache.

- **`cache_results`** (*boolean*) - Post information cache.
- **`update_post_meta_cache`** (*boolean*) - Post meta information cache.
- **`update_post_term_cache`** (*boolean*) - Post term information cache.

Show Posts without adding post information to the cache

Display 50 posts, but don't add post information to the cache:

```
$args = array(
        'posts_per_page' => 50,
        'cache_results'  => false
);
$query = new WP_Query( $args );
```

Show Posts without adding post meta information to the cache

Display 50 posts, but don't add post meta information to the cache:

```
$args = array(
        'posts_per_page'         => 50,
        'update_post_meta_cache' => false
);
$query = new WP_Query( $args );
```

Show Posts without adding post term information to the cache

Display 50 posts, but don't add post term information to the cache:

```
$args = array(
        'posts_per_page'         => 50,
        'update_post_term_cache' => false
);
$query = new WP_Query( $args );
```

In general usage you should not need to use these, adding to the cache is the right thing to do, however they may be useful in specific circumstances. An example of such circumstances might be when using a WP_Query to retrieve a list of post titles and URLs to be displayed, but in which no other information about the post will be used and the taxonomy and meta data won't be needed. By not loading this information, you can save time from the extra unnecessary SQL queries.

Note: If a persistent object cache backend (such as memcached) is used, these flags are set to false by default since there is no need to update the cache every page load when a persistent cache exists.

Return Fields Parameter

Set return values.

- **`fields`** (*string*) - Which fields to return. All fields are returned by default. There are two other options:
  - '`ids`' - Return an array of post IDs.
  - '`id=>parent`' - Return an array of `stdClass` objects with `ID` and `post_parent` properties.
  - Passing anything else will return all fields (default) - an array of post objects.

Filters

- `posts_distinct` - Alters SQL 'DISTINCTROW' clause to the query that returns the post array.
- `posts_groupby` - Alters SQL 'GROUP BY' clause of the query that returns the post array.
- `posts_join` - Alters SQL 'JOIN' clause of the query that returns the post array.
- `post_limits` - Alters SQL 'LIMIT' clause of the query that returns the post array.
- `posts_orderby` - Alters SQL 'ORDER BY' clause of the query that returns the post array.
- `posts_where` - Alters SQL 'WHERE' clause of the query that returns the post array.
- `posts_join_paged` - Alters SQL paging for posts using 'JOIN' clause of the query that returns the post array.
- `posts_where_paged` - Alters SQL paging for posts using 'WHERE' clause of the query that returns the post array.
- **posts_fields** - Alters SQL 'SELECT' clause of the query that returns the post array.
- `posts_clauses` - Alters *all* the SQL clauses above in one go. It gives you an array of elements that are easy to alter (available with Version 3.1).

Note, that there are more filters than the mentioned. As it is hard to keep the codex up to date, please inspect the `get_posts()` method inside the `WP_Query` class yourself (`wp-includes/class-wp-query.php`).

Change Log

- 4.9.0:
  - Introduced the `$comment_count` parameter.

- 4.6.0:
  - Add filter '`posts_pre_query`' to filter the posts array before the query takes place.
  - Add '`post_name__in`' to '`$orderby`'.

- 4.5.0:
  - Removed the '`$comments_popup`' property.

- 4.4.0:
  - Add filter '`old_slug_redirect_url`' to filter the old slug redirect URL.
  - Add the ability to pass a post ID to '`$post`' parameter of '`setup_postdata`' method and function.
  - Introduced '`$post_name__in`' and '`$title`' parameters.
  - '`$s`' was updated to support excluded search terms, by prepending a hyphen.

- 4.1.0:
  - Add '`$this`' parameter to '`the_post`' action (The current Query object (passed by reference)).

Source File

`WP_Query()` is located in `wp-includes/class-wp-query.php`.

External Resources

- WordPress WP_Query Generator
- Advanced Taxonomy Queries with Pretty URLs
- Advanced Taxonomy Queries in WordPress 3.1
- Advanced Metadata Queries in WordPress 3.1
- Comprehensive Argument Reference by Mark Luetke
- WordPress custom loop
- An In-Depth Guide to Conquering WP_Query

Related

Articles

- Article: The Loop - A basic overview of its use of query within the WordPress loop.
- Article: Query Overview - Explanation of how to determine which queries generate WordPress.
- Article: Customizing Queries via Hook
- Article: View Articles MYSQL query using custom
- Article: Build advanced queries on Taxonomies
- Article: Build custom query using Offset and pagination

Code Documentation

- Class: **WP_Query** - Detailed Overview of class WP_Query
- Class: WP_Comment_Query - Class for comment-related queries
- Class: WP_User_Query - Class for user-related queries
- Object: $wpdb - Overview on the use of the $wpdb object
- Function: set_query_var()
- Function: get_query_var()
- Function: query_posts() - Create additional custom query
- Function: get_post() - Take an ID of an item and return the records in the database for that article
- Function: get_posts() - A specialized function that returns an array of items
- Function: get_pages() - A specialized function that returns an array of pages

- Function: have_posts() - A condition that determines whether the query returned an article
- Function: the_post() - Used to automatically set the loop after a query
- Function: rewind_posts() - Clears the current loop
- Function: setup_postdata() - Sets the data for a single query result within a loop
- Function: wp_reset_postdata() - Restores the previous query (usually after a loop within another loop)
- Function: wp_reset_query()
- Function: is_main_query() - Ensures that the query that is being changed is only the main query
- Action Hook: pre_get_posts - Change WordPress queries before they are executed
- Action Hook: the_post - Modify the post object after query
- Filter Hook: found_posts - Changes the value of the object found_posts WP_Query

See also index of Class Reference and index of Function Reference.

This article is marked as in need of editing. You can help Codex by editing it.

Categories:
- Classes
- Copyedit

WordPress.com
Matt
Privacy
Public Code

Follow @WordPress

Like 1.1M