



The Handbook

Codex

http://codex.wordpress.org/The_Loop

Version Date

8 August 2005

The Loop

The Loop is used by WordPress to display each of your posts. Using The Loop, WordPress processes each of the posts to be displayed on the current page and formats them according to how they match specified criteria within The Loop tags. Any [HTML](#) or [PHP](#) code placed between where The Loop starts and where The Loop ends will be used for each post. When WordPress documentation states "This tag must be within The Loop", such as for specific [Template Tag](#) or plugins, this is **The Loop**.

For example, among the information The Loop displays by default in WordPress 1.5 are: the Title ([the_title\(\)](#)), Time ([the_time\(\)](#)), and Categories ([the_category\(\)](#)) for each post. Other information about each post can be displayed with the appropriate [Template Tags](#) or (for advanced users) by accessing the `$post` variable, which is set with the current post's information while The Loop is running.

For a beginners look at The Loop, see [The Loop in Action](#).

Using The Loop

The Loop should be placed in `index.php` and in any other Templates used to display post information, but looks slightly different depending on your version of WordPress. You should first "[find what version of WordPress you have](#)".

WordPress 1.5

Be sure to include the call for the header Template at the top of your [Theme's](#) Templates. If you are using The Loop inside your own design, set `WP_USE_THEMES` to `false`.

```
define('WP_USE_THEMES', false);  
<?php get_header(); ?>
```

The loop starts here:

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
```

and ends here:

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

WordPress 1.2

Be sure to include the call for `wp-blog-header.php` at the top of your index page. Remember, the path for `wp-blog-header.php` must be set to the location of your `wp-blog-header.php` file:

```
<?php /* Don't remove this line. */ require('./wp-blog-header.php'); ?>
```

The loop starts here:

```
<?php if ( $posts ) : foreach ( $posts as $post ) : start_wp(); ?>
```

and ends here:

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

Loop Examples

Style Posts From Some Category Differently

For WordPress v1.5 Only

This example, using the [syntax for version 1.5](#), displays each post with its Title (which is used as a link to the Post's [Permalink](#)), Categories, and Content. It is a simple, bare-bones example; likely your Templates will display more information in a way making things easier to style with [CSS](#).

In order to be a little more instructive, though, this example also allows posts in a category with Category ID '3' to be styled differently. To accomplish this, the [in_category\(\)](#) [Template Tag](#) is used.

The `<!-- -->` tags are HTML comment tags; if you use this example, these tags will not display in web browsers. They serve no purpose other than to annotate the code below.

```
<!-- Start the Loop. -->
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>

<!-- The following tests if the current post is in category 3. -->
<!-- If it is, the div box is given the CSS class "post-cat-three". -->
<!-- Otherwise, the div box will be given the CSS class "post". -->
<?php if ( in_category('3') ) { ?>
    <div class="post-cat-three">
<?php } else { ?>
    <div class="post">
<?php } ?>

<!-- Display the Title as a link to the Post's permalink. -->
<h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>

<!-- Display the Time. -->
<small><?php the_time('F jS, Y'); ?></small>

<!-- Display the Post's Content in a div box. -->
<div class="entry">
    <?php the_content(); ?>
</div>

<!-- Display a comma separated list of the Post's Categories. -->
<p class="postmetadata">Posted in <?php the_category(', '); ?></p>
</div> <!-- closes the first div box -->

<!-- Stop The Loop (but note the "else:" - see next line). -->
```

```

<?php endwhile; else: ?>

<!-- The very first "if" tested to see if there were any Posts to -->
<!-- display. This "else" part tells what do if there weren't any. -->
<p>Sorry, no posts matched your criteria.</p>

<!-- REALLY stop The Loop. -->
<?php endif; ?>

```

Note: Anytime you want to use [HTML](#) code, you *must* be outside the `<?php ?>` tags. [PHP](#) code (even things as simple as curly braces: `{ }`) *must* be inside the `<?php ?>` tags. You can start and stop the PHP code in order to intersperse HTML code even within `if` and `else` statements, as show in the above example.

Exclude Posts From Some Category

For WordPress v1.5 Only

This example can be used to exclude a certain Category from being displayed. It is based on the example [above](#). Since we want to *exclude* a particular category, we again use the [in_category\(\) Template Tag](#), but here we *negate* its output by using PHP's `NOT` operator, the exclamation point: `!`

```

<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>

<!-- The following tests if the current post is in category 3. -->
<!-- If it is not, the code within The Loop is executed as normal. -->
<!-- If it is, nothing is done until the next post is processed. -->
<?php if ( !(in_category('3')) ) { ?>

<div class="post">

    <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>

    <small><?php the_time('F jS, Y'); ?></small>

    <div class="entry">
        <?php the_content(); ?>
    </div>

    <p class="postmetadata">Posted in <?php the_category(', '); ?></p>
</div> <!-- closes the first div box -->

<?php } ?> <!-- Close the if statement. -->

<?php endwhile; else: ?>
<p>Sorry, no posts matched your criteria.</p>
<?php endif; ?>

```

Note: If you use this example for your main page, you should use a different [Template](#) for your Category archives. Otherwise, WordPress will exclude all posts in Category 3 even when viewing that Category Archive!

However, if you want to use the same template file, you can avoid this by using the `is_home()` tag:

```

...
<?php if ( !(in_category('3') && is_home()) ) { ?>
...

```

This will ensure that posts from Category 3 will only be excluded from the main page. There are other [Conditional Tags](#) that can be used to control the output depending on whether or not a particular condition is true with respect to the requested page.

Multiple Loops

This section deals with advanced use of The Loop. It's a bit technical – but don't let that scare you. We'll start off at easy and work up from there. With a little common sense, patience, and enthusiasm, you too can do multiple loops.

First off, “**why would one want to use multiple loops?**” In general, the answer is that you might want to do *something* with one group of posts, and do *something different* to another group of posts, but display both groups on the same page. Something could mean almost anything; you are only limited by your PHP skill and your imagination.

We will get into examples below, but first you should read about the basics. Take a look at the basic Loop. It consists of:

```
<?php if (have_posts()) : ?>
    <?php while (have_posts()) : the_post(); ?>
    <!-- do stuff ... -->
<?php endwhile; ?>
```

In English (PHP types and people familiar with code speak can skip to below): To translate to English, the above would be read: If we are going to be displaying posts, then get them, one at a time. For each post in the list, display it according to `<!-- do stuff ... -->`. When you hit the last post, stop. The `do stuff` line(s), are template dependant.

Aside on `Do stuff`: in this example it is simply a placeholder for a bunch of code that determines how to format and display each post on a page. This code can change depending on how you want your WordPress to look. If you look at the Kubrick theme's `index.php` the `do stuff` section would be everything below:

```
<?php while (have_posts()) : the_post(); ?>
```

To above:

```
<?php comments_popup_link('No Comments »', '1 Comment »', '% Comments »'); ?>
```

An explanation for the coders out there: The `have_posts()` and `the_post()` are convenience wrappers around the global `$wp_query` object, which is where all of the action is. The `$wp_query` is called in the blog header and fed query arguments coming in through `GET` and `PATH_INFO`. The `$wp_query` takes the arguments and builds and executes a DB query that results in an array of posts. This array is stored in the object and also returned back to the blog header where it is stuffed into the global `$posts` array (for backward compatibility with old post loops).

Once WordPress has finished loading the blog header and is descending into the template, we arrive at our post Loop. The `have_posts()` simply calls into `$wp_query->have_posts()` which checks a loop counter to see if there are any posts left in the post array. And `the_post()` calls `$wp_query->the_post()` which advances the loop counter and sets up the global `$post` variable as well as all of the global post data. Once we have exhausted the loop, `have_posts()` will return false and we are done.

Loop Examples

Below are two example of using multiple loops. The key to using multiple loops is that `$wp_query` can only be called once. In order to get around this it is possible to re-use the query by calling `rewind_posts()` or by creating a new query object. This is covered in example 1. In example 2, using a variable to store the results of a query is covered. Example 3 documents the use of `update_post_caches()` ; function to avoid common plugin problems. Finally, 'multiple loops in action' brings a bunch of ideas together to document one way of using multiple loops to promote posts of a certain category on your blog's homepage.

Multiple Loops Example 1

In order to loop through the same query a second time, call `rewind_posts()`. This will reset the loop counter and allow you to do another loop.

```
<?php rewind_posts(); ?>

<?php while (have_posts()) : the_post(); ?>
    <!-- Do stuff... -->
<?php endwhile; ?>
```

If you are finished with the posts in the original query, and you want to use a different query, you can reuse the `$wp_query` object by calling `query_posts()` and then *looping back* through. The `query_posts()` will perform a new query, build a new posts array, and reset the loop counter.

```
// Get the last 10 posts in the special_cat category.
<?php query_posts('category_name=special_cat&showposts=10'); ?>

<?php while (have_posts()) : the_post(); ?>
    <!-- Do special_cat stuff... -->
<?php endwhile;?>
```

If you need to keep the original query around, you can create a new query object.

```
<?php $my_query = new WP_Query('category_name=special_cat&showposts=10'); ?>

<?php while ($my_query->have_posts()) : $my_query->the_post(); ?>
    <!-- Do special_cat stuff... -->
<?php endwhile; ?>
```

The query object `my_query` is used because you cannot use the global `have_posts()` and `the_post()` since they both use `$wp_query`. Instead, call into your new `$my_query` object.

Multiple Loops Example 2

Another version of using multiple Loops takes another tack for getting around the inability to use `have_posts()` and `the_post()`. To solve this, you need to store the original query in a variable, then re-assign it when with the other Loop. This way, you can use all the standard functions that rely on all the globals.

For example:

```
// going off on my own here
```

```

<?php $temp_query = $wp_query; ?>
<!-- Do stuff... -->

<?php query_posts('category_name=special_cat&showposts=10'); ?>

<?php while (have_posts()) : the_post(); ?>
    <!-- Do special_cat stuff... -->
<?php endwhile; ?>

// now back to our regularly scheduled programming
<?php $wp_query = $temp_query; ?>

```

Multiple Loops Example 3 - Plugins

It has been found that some plugins don't play nice with multiple loops. In these cases it was discovered that some plugins which deal with the keyword(s) and tagging of posts, only work for the first instance of a loop in a page where that loop consists of a subset of total posts. If you find that this is the case, you might want to try the following implementation of the basic loop which adds the `update_post_caches($posts)` function. This function resets the post cache and is as yet undocumented. This implementation would be used on the second loop in a page only if the first loop retrieves a subset of posts.

Simply amend:

```

<?php if (have_posts()) : ?>
    <?php while (have_posts()) : the_post(); ?>
    <!-- Do stuff... -->
<?php endwhile; ?>

```

to become:

```

<?php if (have_posts()) : ?>
    <?php while (have_posts()) : the_post();
update_post_caches($posts); ?>
    <!-- Do stuff... -->
<?php endwhile; ?>

```

Multiple Loops in Action

The best way to understand how to use multiple loops is to actually show an example of its use. Perhaps the most common use of multiple loops is to show two (or more) lists of posts on one page. This is often done when a webmaster wants to feature not only the very latest post written, but also posts from a certain category.

Leaving all formatting and CSS issues aside, let us assume we want to have two lists of posts. One which would list the most recent posts (the standard 10 posts most recently added), and another which would contain only one post from the category 'featured'. Posts in the 'featured' category should be shown first, followed by the second listing of posts (the standard). **The catch is that no post should appear in both categories.**

Step 1. Get only one post from the 'featured' category.

```

<?php $my_query = new WP_Query('category_name=featured&showposts=1');
while ($my_query->have_posts()) : $my_query->the_post();
    $do_not_duplicate = $post->ID; ?>

```

```
<!-- Do stuff... -->
<?php endwhile; ?>
```

In English the above code would read:

Set `$my_query` equal to the result of querying all posts where the category is named `frontpage` and by the way, get me one post only. Also, set the variable `$do_not_duplicate` equal to the ID number of the single post returned. Recall that the `Do stuff` line represents all the formatting options associated for the post retrieved.

Note that we will need the value of `$do_not_duplicate` in the next step to ensure that the same post doesn't appear in both lists.

Step 2. The second loop, get the X latest posts (except one).

The following code gets X recent posts (as defined in WordPress preferences) save the one already displayed from the first loop and displays them according to `Do stuff`.

```
<?php if (have_posts()) : while (have_posts()) : the_post();
if( $post->ID == $do_not_duplicate ) continue; update_post_caches($posts); ?>
<!-- Do stuff... -->
<?php endwhile; endif; ?>
```

In English the above code would read:

Get all posts, where a post equals `$do_not_duplicate` then just do nothing (`continue`), otherwise display all the other the posts according to `Do stuff`. Also, update the cache so the tagging and keyword plugins play nice. Recall, `$do_not_duplicate` variable contains the ID of the post already displayed.

The End Result

Here is what the final piece of code looks like without any formatting:

```
<?php $my_query = new WP_Query('category_name=featured&showposts=1');
while ($my_query->have_posts()) : $my_query->the_post();
$do_not_duplicate = $post->ID;?>
<!-- Do stuff... -->
<?php endwhile; ?>
<!-- Do other stuff... -->
<?php if (have_posts()) : while (have_posts()) : the_post();
if( $post->ID == $do_not_duplicate ) continue; update_post_caches($posts); ?>
<!-- Do stuff... -->
<?php endwhile; endif; ?>
```

The end result would be a page with two lists. The first list contains only one post -- the most recent post from the 'feature' category. The second list will contain X recent posts (as defined in WordPress preferences) *except* the post that is already shown in the first list. So, once the feature post is replaced with a new one, the previous feature will show up in standard post list section below (depending on how many posts you choose to display and on the post frequency). This technique (or similar) has been used by many in conjunction with knowledge of the [Template Hierarchy](http://codex.WordPress.org/Template_Hierarchy) (http://codex.WordPress.org/Template_Hierarchy) to create a different look for `home.php` and `index.php`. See associated resources at the bottom of this page.

Sources:

This article on multiple loops is a combination of [Ryan Boren](http://boren.nu) (<http://boren.nu>) and [Alex King's](http://www.alexking.org) (<http://www.alexking.org>) [discussion](http://comox.textdrive.com/pipermail/hackers/2005-January/003578.html) (<http://comox.textdrive.com/pipermail/hackers/2005-January/003578.html>)

about the Loop on the [Hackers Mailing List](#) as well as the tutorial written at [MaxPower](http://www.maxpower.ca/wordpress-hack-sticky-adhesive-kubrick/2005/05/03/) (<http://www.maxpower.ca/wordpress-hack-sticky-adhesive-kubrick/2005/05/03/>).

More Loop Resources

To learn more about the WordPress Loop, and the various template tags that work only within the Loop, here are more resources.

- [The Loop in Action](#)
- [Template Tags](#)
- [Using the Loop in Template Files](#)

External Links

- [Matt Read Loop Article](http://mattread.com/archives/2005/04/wordpress-is-not-php/) (<http://mattread.com/archives/2005/04/wordpress-is-not-php/>)
- [MaxPower Dynamic Sticky Tutorial](http://www.maxpower.ca/wordpress-hack-sticky-adhesive-kubrick/2005/05/03/) (<http://www.maxpower.ca/wordpress-hack-sticky-adhesive-kubrick/2005/05/03/>)
- [IfElse Query_post Redux](http://www.ifelse.co.uk/archives/2005/04/08/query_posts-redux/) (http://www.ifelse.co.uk/archives/2005/04/08/query_posts-redux/)
- [The Loop and Adding Content Outside of It](http://www.optiniche.com/blog/15/the-wordpress-loop-and-adding-content-outside-of-it/) (<http://www.optiniche.com/blog/15/the-wordpress-loop-and-adding-content-outside-of-it/>)
- [1001 WordPression Loops](#)