



The Handbook

Codex

http://codex.wordpress.org/Template_Tags/How_to_Pass_Tag_Parameters

Version Date

9 August 2005

Template Tags/How to Pass Tag Parameters

Introduction

[Template tags](#) are [PHP](#) functions you can embed in your WordPress page templates to provide dynamic blog content. And like PHP functions, many template tags accept arguments, or parameters. Template tag parameters are variables you can use to change a tag's output or otherwise modify its action in some way. Think of parameters as user options or settings that allow you customize how a template tag works.

In regards to parameters, WordPress template tags come in three "flavors." These are described below:

1. [Tags without parameters](#)
2. [Tags with PHP function-style parameters](#)
3. [Tags with query-string-style parameters](#)

Tags without parameters

Some template tags do not have any options, and thus have no parameters you can pass to them.

The template tag [the_author_firstname\(\)](#) is one that accepts no parameters. This tag simply displays the first name of the author for a post. Tags without parameters should have nothing between the tag function's opening and closing brackets (parentheses):

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

Tags with PHP function-style parameters

For template tags that can accept parameters, some require them to be in the [default PHP style](#) (<http://www.php.net/manual/en/functions.arguments.php>). For these, parameters are passed to a tag function by placing one or more values inside the function's parentheses, or brackets.

The [bloginfo\(\)](#) tag accepts one parameter (known as the **show** parameter) that tells it what information about your blog to display:

```
<?php bloginfo('name'); ?>
```

The [wp_title\(\)](#) tag accepts two parameters: the first is the **sep** or separator parameter, and the second the **echo** or display parameter:

```
<?php wp_title(' - ', TRUE); ?>
```

The first is enclosed in single-quotes and the second is not because the first is a *string*, and the second a *boolean* parameter. (See [Types of parameters](#) for information on parameter types and how to use them.)

Important points to keep in mind for PHP function-style parameters:

- Some functions take multiple parameters.

- Multiple parameters are separated by commas.
- **The order of parameters is important!**

When passing parameters to a template tag's function, make sure you specify values for all parameters *up to the last one you wish to modify*, or the tag may not work as expected. For example, the template tag [get_archives\(\)](#) has six parameters:

```
<?php get_archives('type', 'limit', 'format', 'before',
                    'after', show_post_count); ?>
```

To display the archives list the way you want, let's say you only need to modify the third (**format**) and fifth (**after**) parameters. To do this, you also need to make sure to enter default values for the first, second and fourth parameters, as well:

```
<?php get_archives('', '', 'custom', '', '<br />'); ?>
```

Notice the use of single-quotes to denote *empty* parameter values, which *in this case* forces defaults for those specific parameters. Be aware that defaults can be overwritten when passing empty parameters, as is the case of a parameter specifying a string of text, and there's no way to pass an empty boolean value. So check the documentation for a parameter's default, and when one is specified use it as your parameter value (also see [Types of parameters](#) for information on parameter types). The sixth parameter was left off; this is because WordPress uses the default for any remaining parameters left unspecified.

Make sure to follow the documentation for a template tag carefully, and place your parameters in the order the template function expects. Finally, to use the defaults for all parameters in a template tag, use the tag with no parameter values specified:

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

Tags with query-string-style parameters

The last type of template tag makes use of what's called a query-string style to pass parameters to the tag. These provide a convenient 'wrapper' to tags which use the [PHP function parameter style](#) and have a relatively large number of parameters. For example, the template tag [wp_list_cats\(\)](#) is a wrapper to [list_cats\(\)](#), a tag with eighteen parameters!

If you want to set the **exclude** parameter in [list_cats\(\)](#) (seventeenth in the parameter list) and leave the rest at their defaults, you have to do this:

```
<?php list_cats(TRUE, 'All', 'ID', 'asc', '', TRUE, FALSE, FALSE,
TRUE, TRUE, FALSE, '', '', FALSE, '', '', '10,11,12'); ?>
```

Or you can use [wp_list_cats\(\)](#):

```
<?php wp_list_cats('exclude=10,11,12'); ?>
```

So query-string style tags are useful in that they let you change the values of just those parameters you require, without needing to provide values for all or nearly all of them. However, not all PHP function-style template tags have a query-string style equivalent. (Also note that names for tags that accept query-string style parameters usually start with a 'wp_' prefix, such as [wp_list_cats\(\)](#), but check the documentation on a tag to verify its method for accepting parameters, if any.)

The tag [wp_list_authors\(\)](#) has six parameters, of which we set three here:

```
<?php wp_list_authors('show_fullname=1&feed=rss&optioncount=1'); ?>
```

First, all the parameters together are enclosed by either single or double quotes. Then each parameter is entered in the *parameter=value* format, while these are separated with an ampersand (&). Broken down, the tag as shown above states:

- Parameter **show_fullname** (a *boolean* type parameter) equals 1 (true).
AND
- Parameter **feed** (a *string* type parameter) equals `rss`.
AND
- Parameter **optioncount** (a *boolean* type parameter) equals 1 (true).

(See [Types of parameters](#) for information on parameter types and how to use them.)

Parameters in the query-string style do not have to be entered in a specific order. The only real concern is assuring parameter names are spelled correctly. If legibility is a problem, you can separate parameters with a space:

```
<?php wp_list_authors('show_fullname=1 & feed=rss & optioncount=1'); ?>
```

You can also spread a query-string over several lines (note the specific format of enclosing each parameter/value pair in single quotes and a dot starting each new line):

```
<?php wp_list_authors(  
    'show_fullname=1'  
    . '&feed=rss'  
    . '&optioncount=1'  
); ?>
```

There are a few limitations when using query-string style tags, among which you cannot pass certain characters, such as the ampersand or a quote mark (single or double). In those cases, you have to use the PHP function-style tag equivalent.

Types of parameters

There are three types of parameters you need to know about in regards to WordPress template tags: string, integer, and boolean. Each is handled a bit differently, as described below.

String

A string is a line of text, and is typically anything from a single character to several dozen words. A string parameter is often a selection from two or more valid options, as is the **show** parameter in [bloginfo\(\)](#). Otherwise, a string is intended as text to be displayed, such as the **sep** parameter in [wp_title\(\)](#).

In tags which use the [PHP function parameter style](#), string values should be enclosed in single (') or double (") quotation marks. If a single or double quote is required for part of your string, mix the marks (using double quotes to enclose the parameter if a single quote exists in your parameter value), or use the PHP escape character (a backslash: \), as the following does to assign single quotes for the **before** and **after** parameters in [the_title\(\)](#):

```
<?php the_title('\'', '\'); ?>
```

Integer

An integer is a whole number, and can have a positive (1, 2, 3) or negative (-1, -2, -3) value. Integer parameters are often used for date and archive based information, like the **year** and **month** parameters for the [get_month_link\(\)](#) tag, or for specifying the numeric value of something on your blog, as one finds in the case of the **id** parameter in [get_permalink\(\)](#).

When passed to a [PHP function parameter style](#) tag, integer values either in or out of quotes will be handled correctly. So the following examples are both valid:

```
<?php get_permalink('100'); ?>
```

```
<?php get_permalink(100); ?>
```

Boolean

Boolean parameters provide a simple true/false evaluation.

For example, the [the_date\(\)](#) tag has an **echo** parameter which takes either `TRUE` or `FALSE` as a value; setting the parameter to `TRUE` displays the date on the page, whereas `FALSE` causes the tag to "return" the date as a value that can then be used in other PHP code.

A boolean parameter can be notated as a numeric value: 1 for `TRUE`, 0 for `FALSE`. For a boolean value in [PHP function parameter style](#) tags, these are all equivalent:

- 1 = `TRUE` = `true`
- 0 = `FALSE` = `false`

However, do **NOT** enclose boolean values within quote marks. For [query-string style tags](#), use only the numeric boolean values (1 or 0).