<http://docs.presscustomizr.com/article/26-wordpress-actions-filters-and-hooks-a-guide-for-non-developers>

**WordPress Actions, Filters, and Hooks : A guide for non-developers**

When I was learning how to use hooks in WordPress and the Customizr theme, I got tired of searching for articles that explained actions, filters and hooks, in simple terms. Most articles are complex, are written for developers, and start like this:

Actions and filters allow you add your own functionality or modify your site’s behaviour by hooking a callback function onto a specific tag in the core code, setting priorities and using parameters and arguments passed to the callback function.

Whoa!

Don’t worry. *This* isn’t *that* sort of article. If you don’t understand anything in that paragraph, but wish you did, read on….

**The basics**

A WordPress page is made up of a whole load of functions and database queries, with WordPress and the theme working together to output text, images, stylesheets, and other files. The browser interprets all of these and puts them all together into one web page.

Throughout its code, WordPress has included “hooks”, so that people can “hang” their own code on those hooks. A lot of the Customizr theme  [code snippets](http://presscustomizr.com/code-snippets" \t "_blank) are written in PHP, using them.

There are two types of hooks: actions and filters.

* Actions allow you to add extra functionality at a specific point in the processing of the page—for example, you might want to add extra widgets or menus, or add a promotional message to your page.
* Filters allow you to intercept and modify data as it is processed—for example, you might want to insert another CSS class in a WordPress HTML element, or modify some of your pages blocks.

**In brief: actions do stuff; filters change stuff.**

**Getting down to details**

When Gill comes by, tell her to go to the store to get some paint. When Jack comes by bragging about how great he is, get him to say that Gill is better.

Wait, what? I thought this was an article about actions, filters and hooks? Instead, we’re talking about arrogant home-improvers? Well yes. These are ways of understanding what’s going on when you use actions and filters.

Let’s look at actions and filters in turn.

**Actions:**

When Gill comes by, tell her to go to the store to get some paint.

We can do this by hooking onto an action hook. That is, we can use the add\_action function (written add\_action()) to add functionality—do stuff—at a particular point.

Sending Gill to the store to get paint might look like this in PHP:

// Send Gill to get paint

add\_action( 'after\_gill\_arrives' , 'send\_gill\_to\_get\_paint', 10 , 2 );

function send\_gill\_to\_get\_paint( $gill\_has\_keys, $gill\_has\_car ) {

// If $gill\_has\_keys and $gill\_has\_car are both true

if ( $gill\_has\_keys && $gill\_has\_car ) {

echo 'Gill, please go to the store and get some paint. Thank you!';

}

}<br>

**What did we just do there?**

* We watched for a particular thing to happen—Gill having arrived; in programming terms the ‘after\_gill\_arrives’ hook;
* When it happened, we did something—we sent her to the store to get paint; in programming terms we added an action to call the send\_gill\_to\_get\_paint() function, which prints a message on the page;
* We used the $gill\_has\_keys and $gill\_has\_car arguments to perform some basic logic;
* We set other information:
  + The priority of this action: Whether it will run before, or after, other functions attached to the same hook. In this case we set the priority to 10, the default. If we want another function to run before this, we would give the other function a lower value (which means it will be executed first).
  + How many arguments (variables) that the function accepts. These arguments are passed from the action hook to our function. In this case, we set this value to 2.

**How did it work?**

In simple terms, add\_action() tells WordPress to do something when it arrives at the specified do\_action() hook.

But we can only use our add\_action() in the example above if we know that the ‘after\_gill\_arrives’ action hook exists in our theme/plugin. Our theme/plugin needs to have something like this in it:

do\_action( ‘after\_gill\_arrives’ , $gill\_has\_keys = true , $gill\_has\_car = true );

This tells WordPress to create a hook called ‘after\_gill\_arrives’, run any actions that are attached to this hook, and pass the arguments $gill\_has\_keys and $gill\_has\_car to those actions (the 2 arguments we specified above).

**A simple add\_action() example**

Now let’s look at a simple example using the Customizr theme. Let’s add some text after the header:

// Add some text after the header

add\_action( '\_\_after\_header' , 'add\_promotional\_text' );

function add\_promotional\_text() {

// If we're not on the home page, do nothing

if ( !is\_front\_page() )

return;

// Echo the html

echo "<div>Special offer! June only: Free chocolate for everyone!</div>";

}<br>

Inside the Customizr core code, as the very last action in header.php, Customizr has:

do\_action ( '\_\_after\_header' )

This is where our add\_action() is hooking itself.

You can also use remove\_action() to remove actions from their hooks. For example,  [this snippet removes the slider](http://www.presscustomizr.com/snippet/moving-slider-anywhere/" \t "_blank) from one place—using remove\_action() to unhook it from where it’s currently hooked—and adds it somewhere else.

And how do we add the remove\_action() to the page’s code? Yes, that’s right: with an add\_action(). In the case of snippet for moving the slider, we hook our function in the <head> section at the very top of the page, to make sure it’s executed after all the actions are loaded, but before the rest of the page is built. Take a look at that slider snippet to see if it makes more sense to you now.

**Filters:**

When Jack comes by, bragging about how he’s the best, tone down his boasting.

We can change Jack’s boasting comments by hooking onto a filter. That is, we can use add\_filter() to modify functionality—change stuff—at a particular point.

Toning down his boasting might look like this in PHP:

// Cut Jack's boasting

add\_filter( 'jacks\_boast' , 'cut\_the\_boasting');

function cut\_the\_boasting($boast) {

// Replace "best" with "second-best"

$boast = str\_replace ( "best" , "second-best" , $boast );

// Append another phrase at the end of his boast

$boast = $boast . ' However, Gill can outshine me any day.';

return $boast;

}<br>

**What did we just do there?**

* We looked for a particular thing that we wanted to change—Jack’s boast that he’s the best; in programming terms the ‘jacks\_boast’ filter;
* When we found it, we changed it—we changed “best” to “second-best” and we added another phrase on the end; in programming terms, we filtered the output by calling the cut\_the\_boasting() function, which replaced part of the string and appended another string (the “.” character concatenates two PHP strings);
* We used the $boast argument (in this case a string that says something like “I’m the best!”) as the basis for our changes.
* We returned a string at the end of the function. This is very important. If you don’t return a string in a filter, you might disrupt the functioning of a program (in this case, we would simply silence Jack … which may not be a bad thing).

**How did it work?**

In simple terms, add\_filter() tells WordPress to swap its data with ours when it arrives at the ‘jacks\_boast’ filter hook. Neat!

But we can only use our add\_filter() filter in the example above if we know that the ‘jacks\_boast’ filter hook exists in our theme/plugin. (If the hook doesn’t exist, our function will simply never be called.) Behind the scenes, the theme/plugin has to have something like this in it:

echo apply\_filters('jacks\_boast', "I'm the best in the world.");

This tells WordPress to create a hook called ‘jacks\_boast’, apply any filters that are attached to this hook, and pass those filters the string “I’m the best in the world.”. If there are no filters attached to the hook, then the apply\_filters() function will simply return the string “I’m the best in the world.”: effectively a default.

**A simple add\_filter() example**

Now let’s look at an easy example in the Customizr theme. Let’s change the url of the link in the logo:

// Change url that is linked from logo

add\_filter( 'tc\_logo\_link\_url', 'change\_site\_main\_link' );

function change\_site\_main\_link() {

return 'http://example.com';

}<br>

Inside the Customizr core code, in the function that displays the logo (in class-header-header\_main.php), Customizr has:

apply\_filters( ‘tc\_logo\_link\_url’, esc\_url( home\_url( ‘/’ ) ) )

This is where our add\_filter() is hooking itself. The esc\_url() function eliminates invalid characters etc. in urls and the home\_url() function retrieves the home url for the site. So without any filtering, the ‘tc\_logo\_link\_url’ filter returns the home page’s address.

In this example, we didn’t even take any notice of the incoming arguments (the home url), because we knew we were just going to completely overwrite it.

Remember: When you use a filter, you must always return something.

**Why use hooks?**

Now you know how they work, you can see that understanding hooks is absolutely necessary for anyone developing with WordPress. It’s also very useful even if you are not a developer but want to modify WordPress’s—or your theme’s— behaviour.

With an understanding of hooks, you can:

* **Change almost anything in WordPress—even quite fundamental things—because a lot of WordPress’s core functions use actions and filters;**
* Make changes easily: once you’ve understood the concepts, you can make some incredibly complex changes very quickly;
* Change a theme's behaviour at the source, rather than trying to retro-fit an inappropriate solution with HTML and CSS;
* Make your own changes easy to understand and easier to debug, because your code is reduced to a minimum;
* Enable and disable your changes easily because each piece of code is a small unit in your functions.php;
* Make your changes relatively upgrade-proof because you no longer need to edit or copy WordPress or any themes and plugins core files;
* Share your knowledge and swap code snippets with others.

Fully understanding hooks can take a few hours or (much) longer, depending on your skills, but it will save you days of time in the future.

**Try it yourself**

All the above examples are working examples. If you have a test installation of WordPress to play with, you can try using the code above in the functions.php in your child theme. Some things to bear in mind:

* In the add\_action() example that echoes text for Gill:
  + You will need to include the line do\_action( ‘after\_gill\_arrives’ , $gill\_has\_keys = true , $gill\_has\_car = true ); in your code, because this action hook isn’t in Customizr.
  + If both $gill\_has\_keys and $gill\_has\_car are set to true, then your website will display “Gill, please go to the store and get some paint. Thank you!” at the top of the page. If you change either one or both of these values to false, the message will not display.
* In the add\_action() example to add promotional text: Don’t include the ‘\_\_after\_header’ hook in your code, because this already included in Customizr’s core code.
* In the add\_filter() example that changes Jack’s words:
  + You will need to include line echo apply\_filters(‘jacks\_boast’,”I’m the best in the world.”); in your code, because this filter hook isn’t in Customizr.
  + Your website will display “I’m the second-best in the world. However, Gill can outshine me any day.” at the top of the page.
* In the add\_filter() example to change the logo url: Don’t include the ‘tc\_logo\_link\_url’ filter hook in your code, because this already included in Customizr’s core code.

**Where to copy/paste this code?**

In your functions.php file. We strongly recommend you create a  [child theme.](https://codex.wordpress.org/Child_Themes" \t "_blank)

Remember: you shouldn’t edit the theme’s functions.php, nor should you put a copy of it into your child theme.

**Where next?**

Take a look at WordPress code, or any themes or plugins code, to see where there are do\_action() and apply\_filters() functions. You can search using your favourite desktop search tool. It will give you an idea of the variety of things you can customize.

Take a look at the following to extend your knowledge further:

* A really great diagram showing the [anatomy of a WordPress page](http://www.rarst.net/script/wordpress-core-load/), before the theme gets to weave its magic.
* WordPress Codex: [add\_action](http://codex.wordpress.org/Function_Reference/add_action), [do\_action](http://codex.wordpress.org/Function%20Reference/do_action), [remove\_action](http://codex.wordpress.org/Function%20Reference/remove_action), [add\_filter](http://codex.wordpress.org/Function%20Reference/add_filter), [apply\_filters](http://codex.wordpress.org/Function%20Reference/apply_filters)
* Customizr [code snippets using add\_action](http://www.presscustomizr.com/function/add_action/)
* Customizr [code snippets using add\_filter](http://www.presscustomizr.com/function/add_filter/)

Let us know how you get on!

credits :  [@electricfeet](https://profiles.wordpress.org/ElectricFeet)