central
INSTITUTE OF TECHNOLOGY

Perth | Western Australia

WordPress Child Themes Part 2

Version 1.0
Guido Verschoor

updated 11th June 2015

Contents

References:

reference for this PDF:
1) http://themeshaper.com/2009/04/30/modular-css-wordpress-child-themes/
2) http://themeshaper.com/2009/05/03/filters-wordpress-child-themes/
3) http://themeshaper.com/modify-wordpress-themes/
4) http://premium.wpmudev.org/blog/how-to-create-wordpress-child-theme/?utm_expid=3606929-46.O3zA8dIfQ7OjzdOy6sId8g.0&utm_referrer=https%3A%2F%2Fwww.google.com.au

Resources

Theme Development
How to Modify WordPress Themes the Smart Way (four part series)
How To Make Your Own Child Theme - Includes Screencast
Guide to WordPress Child Theme Development
How to: Create a Child Theme based on Twenty Eleven
Customizing your WordPress theme using Firebug
Tutorial: Child Themes basics and creating Child Themes in WordPress
How to modify the Parent Theme behavior within the Child Theme
Child Theme Plugins
WordPress Child Theme The Why and How

Extra reading - gives actual examples of style.css and function.php pages:
How to create Twentythirteen child theme for WordPress

Theme Downloads (will add more soon):
Extra Themes
Big Theme

references for hooks, actions and filters:
Customising 2015 theme
https://codex.wordpress.org/Plugin_API#Hooks.2C_Actions_and_Filters
http://codex.wordpress.org/Glossary#Filter
http://codex.wordpress.org/Glossary#Hook
http://codex.wordpress.org/Glossary#Action

Child Theme Mechanics

So how does a child theme actually work? Child themes work on a file-level. When a file is used during the process of loading a theme it checks if it is present in the child theme. If it is, the content of that file is used. If it isn't, the same file in the parent theme is used.

There is one exception to this rule, the theme's functions file. The **functions.php** file in both the parent and the child theme is loaded. If the child theme's functions replaced the parents you would either have a malfunctioning site, or you would need to copy-paste the entire contents of the parent theme's function file into the child theme's which would sort of defeat the purpose of extending a theme.

The workflow when modifying functionality is the following. If you want to make changes to the header, copy-paste the parent theme's header.php file into your child theme. Edit the file to your heart's content, save it and enjoy the fruits of your labour.

Some Notes For Theme Makers

If you make your own themes there are a couple of guidelines you may want to follow to make child theme creation easier. The two most important ones are learning the difference between `get_stylehseet_directory()` and `get_template_directory()` and creating pluggable functions.

What is function.php

One way to change the default behaviors of WordPress is using a file named functions.php. It goes in your Theme's folder.

The functions file behaves like a WordPress Plugin, adding features and functionality to a WordPress site. You can use it to call functions, both PHP and built-in WordPress, and to define your own functions. You can produce the same results by adding code to a WordPress Plugin or through the WordPress Theme functions file.

There are differences between the two.

A WordPress Plugin:

- Requires specific, unique Header text.
- Is stored in wp-content/plugins, usually in a subdirectory.
- Executes only when individually activated, via the Plugins panel.
- Applies to all themes.
- Should have a single purpose, e.g., convert posts to Pages, offer search engine optimization features, or help with backups.

A functions file:

- Requires no unique Header text.
- Is stored with each Theme in the Theme's subdirectory in wp-content/themes.
- Executes only when in the currently activated theme's directory.
- Applies only to that theme. If the Theme is changed, the functionality is lost.
- Can have numerous blocks of code used for many different purposes.

Each theme has its own functions file, but only the **functions.php** in the active Theme affects how your site publicly displays. If your theme already has a functions file, you can add code to it. If not, you can create a plain-text file named **functions.php** to add to your theme's directory.

A Child Theme can have its own functions.php. This child functions file can be used to augment or replace the parent theme's functions.

With a functions file you can:

- Use WordPress **Hooks**, that vast collection of WordPress **actions** and **filters** that can alter almost everything WordPress does. For example, with the excerpt_length filter you can change your Post Excerpt length (from default of 55 words).
- Enable WordPress features such as `add_theme_support()` to turn on Post Thumbnails, Post Formats, and Navigation Menus.
- Define functions you wish to re-use in multiple theme template files.

Beware: if a WordPress Plugin calls the same function, or filter, as you do in your functions file, the results can be unexpected -- even site-disabling.

Search the web for "WordPress functions.php" to find suggestions to enhance the functionality of your WordPress site.

Hooks, Actions and Filters

Hooks are provided by WordPress to allow your plugin to 'hook into' the rest of WordPress; that is, to call functions in your plugin at specific times, and thereby set your plugin in motion. There are two kinds of hooks:

- **Actions**
- **Filters**

You can sometimes accomplish the same goal with either an action or a filter. For example, if you want your plugin to change the text of a post, you might add an action function to publish_post (so the post is modified as it is saved to the database), or a filter function to the_content (so the post is modified as it is displayed in the browser screen).

Action

In WordPress; an Action is a PHP function that is executed at specific points throughout the WordPress Core.

Developers can create a custom Action using the Action API to add or remove code from an existing Action by specifying any existing Hook. This process is called "**hooking**".

For example: A developer may want to add code to the footer of a Theme. This could be accomplished by writing new function, then Hooking it to the `wp_footer` Action.

Filter

In WordPress, a Filter is a function that is associated with an existing Action by specifying any existing Hook.

Developers can create custom Filters using the Filter API to replace code from an existing Action. This process is called "**hooking**".

Custom Filters differ from custom Actions because custom Actions allow you to add or remove code from existing Actions. Whereas custom Filters allow you to replace specific data (such as a variable) found within an existing Action.

Using Filter Hooks in WordPress Child Themes

How WordPress Filter Hooks Work

Read the following first: http://www.kathyisawesome.com/thematic-hooks-and-functions/

Here are most of the Action Hooks as per WordPress documentation: http://codex.wordpress.org/Function_Reference/add_action

Action hooks look like this:

```
?
do_action('this_is_the_action_hook_name`);
```

Kathy from WooCommerce said: Filtering a function is actually pretty easy—once you know how WordPress filters work. But if you're going to filter something in your Child Theme functions you're going to have to get your hands messy with a little PHP. Luckily, it's pretty easy stuff.

If you're just getting started messing around with PHP I want you to remember this one crucial tip: the functions file must begin and end with the opening and closing PHP tags. With no extra lines before or after. Something like this:

```
<?php
// We`re going to be pasting our code between those PHP tags above and below this comment
?>
```

The Anatomy of a WordPress Filter Hook

Let's look at the anatomy of a WordPress Filter Hook.

```
// We`ll use this form if we`re going to just outright replace something
function my_function_name() {
  // Your custom code goes here, between the curly braces
}
add_filter('filter_name`,`my_function_name`);
```


Take a look at the above example. We just wrote some PHP code. We wrote a function:

```
function my_function_name() {
    // Your custom code goes here, between the curly braces
}
```

And then we told WordPress to take that function and use my_function_name instead of the original content, or code, that was returned by filter_name.

```
add_filter('filter_name`,`my_function_name`);
```

filter_name is the key here. Somewhere in the bowels of WordPress, or our Parent Theme code, there is a function that looks something like this:

```
function super_stuff() {
    $stuff = 'bacon`;
    echo apply_filters ( 'filter_name` , $stuff );
}
```

Did you catch that last line in the super_stuff function? echo essentially means "put it on the screen" but the really important bit here is `apply_filters`. That's where we're getting our filter_name, the filter we're, um, well, filtering. Any function using `apply_filters`, whether it's in the WordPress code or your Parent Theme code, will let itself be filtered by another function in a plugin or, more importantly to us here, a WordPress Child Theme.

Let's get to it and start filtering some Theme functions.

Filtering Theme Functions

This example uses a child theme which is no longer supported. I have included this here to demonstrate how filtering a theme function works.

Activate the Child Theme, Chiron, we made in Modular CSS in WordPress Child Themes and make a functions file if you haven't already (chiron/functions.php). We're going to start out by filtering with one of the more complicated looking Theme functions in our Parent Theme, Thematic, thematic_postheader.

Actually it's not that complicated at all. Let's have a look at an abridged version of thematic_postheader that contains almost everything you'll need to know.

```
// Information in Post Header
// Basically the stuff you see at the top of every post
function thematic_postheader() {
    global $id, $post, $authordata;

    // The Post Title
    $posttitle = apply_filters('thematic_postheader_posttitle`,$posttitle);

    // The Post Meta
    $postmeta = apply_filters('thematic_postheader_postmeta`,$postmeta);

    // Is this a post or a page?
    if ($post->post_type == 'page` || is_404()) {
        // If it`s a page show only the Post Title
        $postheader = $posttitle;
    } else {
```

```
        // If it`s a post show the Post Title and The Post Meta
        $postheader = $posttitle . $postmeta;
    }

    // Echo the Post Header
    echo apply_filters( 'thematic_postheader`, $postheader ); // Filter to override default
post header
}
```

Did you look at the code? There's 3 apply_filters in there, 1 for thematic_postheader itself and 2 for the content inside of it, the Post Title and the Post Meta.

Alright. Let's do something stupid and replace everything in our Post Headers with … bacon. Copy this code snippet to your Child Theme functions file, save and reload your test site.

```
function childtheme_postheader() {
    echo 'bacon`;
}
add_filter('thematic_postheader`,`childtheme_postheader`);
```

Your Post Titles and Post Meta should have all disappeared and been replaced with … bacon. Not especially useful but you just learned how to filter the output of any filterable function.

Let's try another—more useful—function. Delete the previous snippet from your functions file.

At some point you might want to add a containing div to all the Post Titles in your theme. Perhaps you want to add a complicated background effect or need to float something in a particular way. We're going to filter the variable in thematic_postheader that's returning the Post Title and make that change to the code output by the Theme template files—without actually editing any template files.

Copy this code snippet to your Child Theme functions file, save and reload your test site.

```
function childtheme_posttitle($posttitle) {
    return '<div class="containing">` . $posttitle . '</div>`;
}
add_filter('thematic_postheader_posttitle`,`childtheme_posttitle`);
```

Check out your test site source code. You just added a containing div to every Post Title on your site. With only 4 lines of code.

Take note of what we did differently in this function. We added a variable in line 1 and returned it in line 2. Doing this lets you use the content of the original function you're filtering. This is powerful stuff.

You've now been armed with enough information to go out and start filtering Parent Theme functions in your WordPress Child Themes (and you're halfway to becoming a WordPress Plugin author too). Now all you need are some functions to filter.

Where To Look For Filters

Besides digging through the code there are two good resources for finding where all the filters you need are hiding.

- The Thematic Theme Guide Filters List
- WordPress Codex Filters List

Using Action Hooks in WordPress Child Themes

In this post we'll review how to write a PHP function and go over the basic idea of how you can use Action Hooks in your WordPress Theme. We'll take a look at a practical example of injecting a Welcome Blurb into your Theme without touching the existing code and we'll also look at how to remove existing content being injected into Theme Hooks.

Packing Up A Function

Action hooks are in a lot of WordPress Themes nowadays. There's a good reason for that but you're probably wondering what the big deal is right? They're such a big deal because firstly, they're incredibly easy to use and secondly, because they're extremely powerful.

If you want to get started with them we're going to have to take a look at how to write a PHP function again. Don't worry, we'll keep it pretty simple.

```
function pretty_basic() {

   // This is a PHP comment.

   // PHP "stuff" goes here.

}

function still_pretty_basic() { ?>

   <!-- HTML comment now -->

   <!-- Notice how I "broke out" of writing PHP? -->

   <!-- I added closing and opening PHP tags just inside the curly braces -->

   <?php // I can even write more familiar-looking PHP here ?>

<?php }
```

So that's how you write a PHP function. It's pretty easy. It's a little package of "stuff" like more PHP or HTML that you can write in 1 place—like your **functions.php** file—and call in another place like so:

```
<?php pretty_basic() ?>

<?php still_pretty_basic() ?>
```

You've seen the same thing before with WordPress functions like wp_list_pages() or the_content(). These functions are packed with "stuff" deep in the bowels of the WordPress core and output wherever they appear in your template files.

The Basic Idea Behind Action Hooks

A lot of really smart theme developers have started adding what are essentially empty functions to their themes ready to be filled up with stuff. We call these Action Hooks. I first noticed them in the Tarski theme and Tarski theme developer Benedict Eastaugh does a really good job of explaining why you'd want to use them and how to add them to your theme. But you've probably already seen them before.

WordPress Themes use a pair of default hooks called wp_head() and wp_footer(). If you know what those are you know what we're doing here. Those two function calls are for adding "stuff" to your WordPress theme without editing the template.

In a nutshell here's what Action Hooks will mean to you if your favorite theme uses them: you can add any content you like—extra WordPress functions, plugin calls, singing and dancing jQuery-powered sliders—simply by adding your function to an existing action hook. You won't have to edit the original template files. And your additions will be safe from any upgrades the theme author makes.

Adding Content To An Action Hook

Let's do something practical we'll add a "Welcome To My Blog" blurb just below the header of the Thematic Theme using the Thematic Action Hook, thematic_belowheader(). Place the following code snippet in your Child Theme functions.php file.

```
// First we make our function
function childtheme_welcome_blurb() {

// We`ll show it only on the HOME page IF it`s NOT paged
// (so, not page 2,3,etc.)
if (is_home() &amp;amp; !is_paged()) { ?>

<!-- our welcome blurb starts here -->
<div id="welcome-blurb">
<p>Welcome to <?php bloginfo('name`); ?>.</p>
</div>
<!-- our welcome blurb ends here -->
<?php }

} // end of our new function childtheme_welcome_blurb

// Now we add our new function to our Thematic Action Hook
add_action('thematic_belowheader`,`childtheme_welcome_blurb`);
```

Make sure you read the comments in the code snippet. It's easy as cake! Want to add something to a WordPress Theme without touching the files and making a headache for yourself when it comes time to upgrade? Just find out where the Action is!

Removing Existing Content From Actions

In some themes, like Thematic, some of the Action Hooks are already partly filled up with stuff. This might look a little unusual if you've never seen it before but it's insanely powerful. You can basically unplug parts of your theme.

Here's the basic syntax for removing content from your Theme that's already being hooked-in.

```
// Unhook default Thematic functions
function unhook_thematic_functions() {
    // Don`t forget the position number if the original function has one
    remove_action('thematic_hook_name`,`thematic_function_name`,postitionnumber);
}
add_action('init`,`unhook_thematic_functions`);
```

And as an example, here's a code snippet that will remove the entire main navigation menu from your Thematic Theme. Just pop it into your Child Theme functions.php and let it rip.


```
// Unhook default Thematic functions
function unhook_thematic_functions() {
    // Don`t forget the position number if the original function has one
    remove_action('thematic_header`,`thematic_access`,9);
}
add_action('init`,`unhook_thematic_functions`);
```

Combine these 2 concepts with a hooked-up WordPress Theme and you can do practically anything.

Practical exercise.

Here is a great webpage which tells you how to modify the 2015 theme in WordPress. Half down the page he explains the use of Action Hooks:
http://premium.wpmudev.org/blog/customize-twenty-fifteen/?utm_expid=3606929-46.O3zA8dlfQ7OjzdOy6sld8g.0&utm_referrer=https%3A%2F%2Fwww.google.com.au