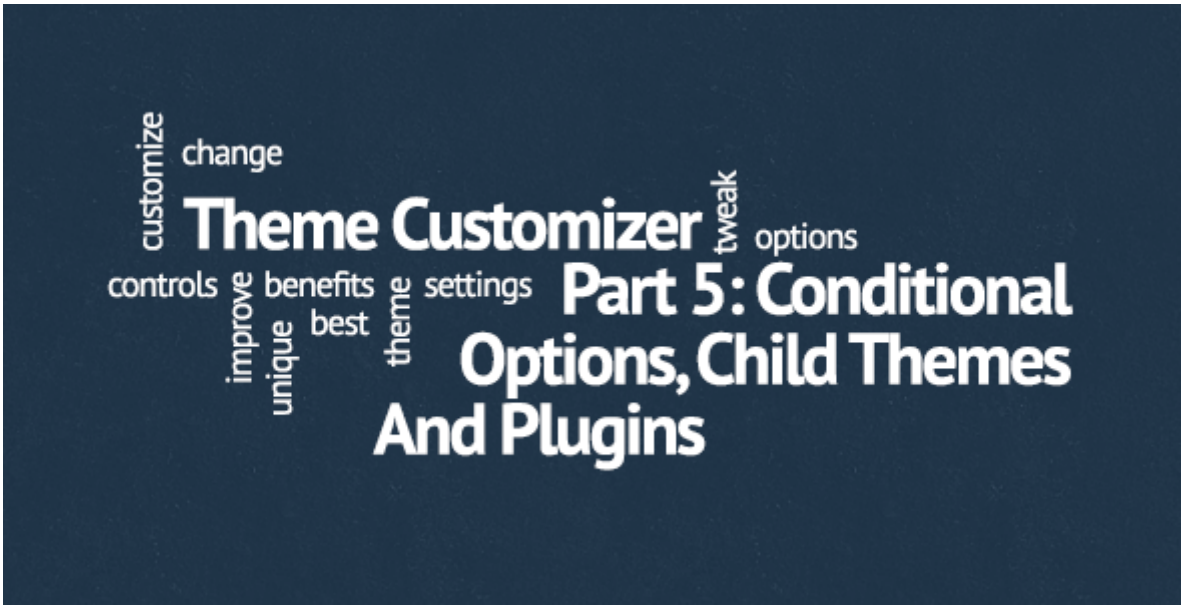
TUTORIALS

Theme Customizer Boilerplate – Conditional Options, Child Themes and Plugins

LAST UPDATED ON: JUNE 26, 2014



So far we've seen how simple it is to handle theme options using Theme Customizer Boilerplate and its hooks. As you probably recall, the most important step was hooking into **'thsp_cbp_options_array'** filter hook and passing it array of options you want to use in your theme.

I'm sure you're already familiar with WordPress action and filter hooks — Plugin API — and how they work, but just in case, here's a quick recap (using filter hooks as an example). You can define your custom function and hook it into an existing filter using add_filter function:

```
add_filter( $tag, $function_to_add, $priority, $accepted_args );
```

Let's focus on priority argument. Its default value is 10, so if you do not use another number, that's what your function's execution priority will be. Lower the number, earlier your function is executed. So if you do something like this:

```
// Adding first message
function my_theme_add_first_message( $content ) {
    $content .= '<p>First Message</p>';
    return $content;
}
add_filter( 'the_content', 'my_theme_add_first_message', 1 );

// Adding second message
function my_theme_add_second_message( $content ) {
    $content .= '<p>Second Message</p>';
    return $content;
}
add_filter( 'the_content', 'my_theme_add_second_message', 2 );
```

When you call the_content function in single.php or any other template post content will be shown, followed by First Message, followed by Second Message. Not because that's their order in this code snippet, but because of execution priority parameter. Think of hooks as if they were snowballs rolling down the hill picking all sort of stuff on their way.

How does this apply to Theme Customizer Boilerplate?

You can hook into **'thsp_cbp_options_array'** from your theme's function.php file, using a custom function (e.g. my_theme_options_array) with priority value set to 1. That means any other function that hooks into **'thsp_cbp_options_array'** filter hook will do it AFTER my_theme_options_array function you already defined. Take a look at this example:

```php
function my_theme_options_array() {
    // Using helper function to get default required capability
    $thsp_cbp_capability = thsp_cbp_capability();

    $options = array(
        // Section ID
        'my_theme_new_section' => array(

            'existing_section' => false,
            'args' => array(
                'title' => __( 'New Section', 'my_theme_textdomain' ),
                'priority' => 10
            ),
            'fields' => array(
                /*
                 * Radio field
                 */
                'my_radio_button' => array(
                    'setting_args' => array(
                        'default' => 'option-2',
                        'type' => 'option',
                        'capability' => $thsp_cbp_capability,
                        'transport' => 'refresh',
                    ),
                    'control_args' => array(
                        'label' => __( 'My Radio Button', 'my_theme_text
                        'type' => 'radio', // Radio control
                        'choices' => array(
                            'option-1' => array(
                                'label' => __( 'Option 1', 'my_theme_tex
                            ),
                            'option-2' => array(
                                'label' => __( 'Option 2', 'my_theme_tex
                            ),
                            'option-3' => array(
                                'label' => __( 'Option 3', 'my_theme_tex
                            )
                        ),
                        'priority' => 3
                    )
                )
            )
        )
    );

    return $options;
}
add_filter( 'thsp_cbp_options_array', 'my_theme_options_array', 1 );
```

This will add New Section to Theme Customizer with one field in it, called My Radio Button. Then you, or someone else develops a child theme for your theme and decides to keep New Section, but instead of My Radio Button it might be better to have My Checkbox. Easy:

```
function my_child_theme_options_array( $options ) {
    // Using helper function to get default required capability
    $thsp_cbp_capability = thsp_cbp_capability();

    /*
     * This time, we're only editing fields in my_theme_new_section in tl
     */
    $options['my_theme_new_section']['fields'] = array(
        'my_checkbox_field' => array(
            'setting_args' => array(
                'default' => true,
                'type' => 'option',
                'capability' => $thsp_cbp_capability,
                'transport' => 'refresh',
            ),
            'control_args' => array(
                'label' => __( 'My Checkbox', 'my_theme_textdomain' ),
                'type' => 'checkbox', // Checkbox field control
                'priority' => 2
            )
        )
    );

    return $options;
}
add_filter( 'thsp_cbp_options_array', 'my_child_theme_options_array', 2 )
```

Noticed that I didn't pass $options parameter to my_theme_options_array and did it in my_child_theme_options_array function? That's because when I first hooked into **'thsp_cbp_options_array'** hook I wanted to override Theme Customizer Boilerplate sample options. Then, when I hooked into it again from my child theme, I didn't want to completely delete parent theme's options, just slightly edit them. That's why I'm only messing with $options['my_theme_new_section']['fields'], not the entire $options array.

Of course, you can also hook into **'thsp_cbp_options_array'** filter hook from your parent theme more than once.. Let's say you chose not to add plugin-territory features to your theme and let plugins do what they're supposed to.

Now you want to show some Theme Customizer options only if a certain plugin is active. Again, easy:

```php
function my_plugin_dependency_options_array( $options ) {
    // Using helper function to get default required capability
    $thsp_cbp_capability = thsp_cbp_capability();

    /*
     * Only adding my_plugin_dependency_section if 'test-plugin.php' is
     */
    if ( is_plugin_active( 'test-plugin/test-plugin.php' ) ) {

        $options['my_plugin_dependency_section'] = array(
            'existing_section' => false,
            'args' => array(
                'title' => __( 'Plugin Dependency', 'my_theme_textdomain
                'priority' => 10
            ),
            'fields' => array(
                /*
                 * Text field
                 */
                // Field ID
                'new_text_field' => array(
                    'setting_args' => array(
                        'default' => __( '', 'my_theme_textdomain' ),
                        'type' => 'option',
                        'capability' => $thsp_cbp_capability,
                        'transport' => 'refresh',
                    ),
                    'control_args' => array(
                        'label' => __( 'Only shows if', 'my_theme_textdo
                        'type' => 'text', // Text field control
                        'priority' => 5
                    )
                ),
            )
        );

    }

    return $options;
}
add_filter( 'thsp_cbp_options_array', 'my_plugin_dependency_options_arra
```

Want to develop a [core functionality plugin] to be used with your theme (as you should)? You can hook into **'thsp_cbp_options_array'** from one of your

plugin's files too, the same way you'd do it from a theme's function.php file.

Don't Go Option Crazy

Every time you're adding options to a theme you develop you need to keep one of WordPress' core principles — **Decision not Options** — in mind. It is easy to get carried away and start adding user options for every minor detail your theme has, but that's not doing anyone a favor. I hope these few tricks, especially adding plugin dependant options, will help keep your theme's options count as low as possible.

After all, if your theme has options for things like every border radius of every single element, it's not a theme it's a WYSIWYG editor and probably not a great one.

You don't buy a white shirt because with some extra effort you can transform it into a table cloth, you buy it because you like its "whiteshirtness". WordPress themes should be like that, too, they should present content in a certain way, not try to do everything in every way imaginable. If you're a theme developer it's your job to make sure user expectations are what they should be.

Article by Slobodan Manic

WPExplorer.com author