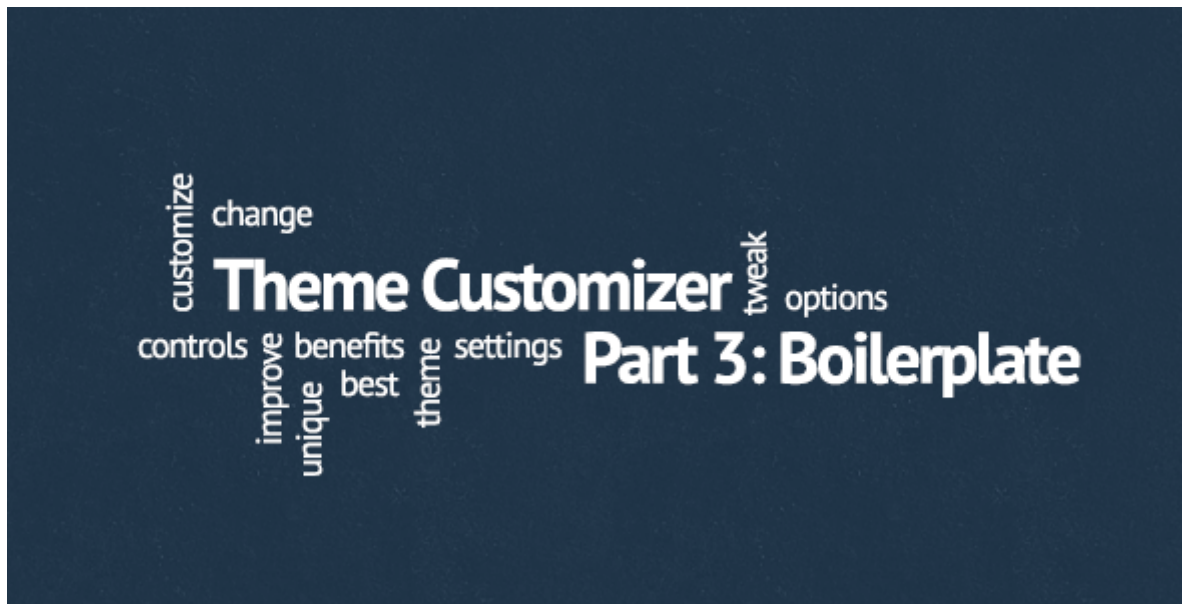


TUTORIALS

WordPress Theme Customizer Boilerplate

LAST UPDATED ON: MAY 6, 2017




1. [Introduction To The WordPress Theme Customizer](#)
2. [Interacting With WordPress Theme Customizer](#)
3. **Currently Reading:** ~~[WordPress Theme Customizer Boilerplate](#)~~
4. [Extending The WordPress Theme Customizer Boilerplate](#)
5. [Theme Customizer Boilerplate – Conditional Options, Child Themes and Plugins](#)






Update: This article has been edited on February 19th 2013, to reflect the changes made to Theme Customizer Boilerplate.

Hopefully you read and enjoyed first two posts of Theme Customizer series — [Introduction to WordPress Theme Customizer](#) and [Interacting With Theme Customizer](#). Now it's time to move to main course and start assembling Theme Customizer boilerplate you'll be able to use in your themes. This post contains a few long blocks of code, so pay attention to inline comments.

Note: If you'd rather just use the boilerplate right away, you can download it from Github and change fields in \$options array by hooking into 'thsp_cbp_options_array' filter hook.

What We're Creating

 slobodan authored 4 hours ago

..		
 custom-controls.php	5 hours ago	Added thsp_cbp_custom_controls action h
 customizer-controls.css	5 hours ago	Added thsp_cbp_custom_controls action h
 customizer.php	4 days ago	Added default sanitize function [slobodan]
 helpers.php	4 hours ago	Renamed a variable, added textarea to sar
 options.php	4 hours ago	Renamed a variable, added textarea to sar

Theme Customizer Boilerplate directory structure

- **customizer.php** – This is the main Theme Customizer boilerplate file, the one that adds sections, settings and controls using data from options array
- **custom-controls.php** – Custom controls classes and an action hook that allows you to add your own custom controls
- **helpers.php** – Helper functions, used to retrieve theme options, option defaults etc.
- **options.php** – Sample options and a filter hook that allows you to edit the options array and use your own fields
- **customizer-controls.css** – Basic CSS for image replaced radio custom control

The whole idea is to be able to copy these files to a subdirectory in your theme directory, include customizer.php file from your functions.php and change anything you like, including and especially the options array, by using Theme Customizer Boilerplate hooks (explained in Part 4). **Update:** Previously, you'd just edit options.php, but using hooks makes things a lot cleaner.

Now let's use a real example — we'll add a text control to a new Theme Customizer section. The array is placed in a helper function and has a filter applied to it when returned. This way you can add more options from a child theme or plugin. Here it is:

```

1  /**
2   * Helper function that holds array of theme options.
3   *
4   * @return array $options Array of theme options
5   * @uses thsp_get_theme_customizer_fields() defined in customizer/he
6   */
7  function thsp_get_theme_customizer_fields() {
8
9      /*
10     * Using helper function to get default required capability
11     */
12     $required_capability = thsp_settings_page_capability();
13
14     $options = array(
15
16         // Section ID
17         'new_customizer_section' => array(
18
19             /*
20             * We're checking if this is an existing section
21             * or a new one that needs to be registered
22             */
23             'existing_section' => false,
24             /*
25             * Section related arguments
26             * Codex - http://codex.wordpress.org/Class_Reference/WP_Cus
27             */
28             'args' => array(
29                 'title' => __( 'New Section Title', 'my_theme_textdomain'
30                 'description' => __( 'New section description', 'my_them
31                 'priority' => 10
32             ),
33
34             /*
35             * 'fields' array contains all the fields that need to be
36             * added to this section
37             */
38             'fields' => array(
39
40                 /*
41                 * =====
42                 * =====
43                 * Text field
44                 * =====
45                 * =====
46                 */
47                 // Field ID
48                 'new_text_field' => array(

```

```

50      /*
51      * Setting related arguments
52      * Codex - http://codex.wordpress.org/Class_Reference/customize_register
53      */
54      'setting_args' => array(
55          'default' => __( 'Default text value', 'my_theme' ),
56          'type' => 'option',
57          'capability' => $required_capability,
58          'transport' => 'refresh',
59          'sanitize_callback' => 'thsp_sanitize_cb',
60      ),
61      /*
62      * Control related arguments
63      * Codex - http://codex.wordpress.org/Class_Reference/customize_register
64      */
65      'control_args' => array(
66          'label' => __( 'New text control label', 'my_theme' ),
67          'type' => 'text', // Text field control
68          'priority' => 1
69      )
70  )
71
72  )
73
74  ),
75
76  );
77
78  /*
79  * 'thsp_customizer_options' filter hook will allow you to
80  * add/remove some of these options from a child theme
81  */
82  return apply_filters( 'thsp_customizer_options', $options );
83
84  }

```

Update: The array stays the same, but now you can also pass the options array to a filter hook, see Part 4 for more details.

Looks complicated at first sight, I know, but let me explain.

The **\$options** array consists of section(s) (only one in this case – `new_customize_section`), each section has it's arguments, fields and a boolean value (`existing_section`) to indicate if it's a new section, or we're just adding some fields to an existing one. **Arguments array is identical to what you'd**

pass to [\\$wp_customize->add_section](#) method. Fields array is slightly more complex.

Update: choices array in control arguments is now a multi-dimensional array.

Each field consists of a Customizer setting and a Customizer control. That's why we have `setting_args` and `control_args` arrays. **They are almost exactly the same as arguments arrays you'd pass to [\\$wp_customize->add_setting](#) and [\\$wp_customize->add_control](#) methods.** The only difference is 'choices' array in control arguments, `$wp_customize->add_control` requires it to be a simple key => value pair array and we're using an multi-dimensional array instead.

This way it's possible to pass more data to each one of the choices, so if you're, for example, loading Google Fonts in your theme, you'll be able to have strings that allow you to load the correct font inside choices array. You can see an example of this [theme that uses Customizer Boilerplate](#).

So, if you're already aware of those three methods, it's all very familiar.

Adding a checkbox control is almost the same, you just need to change 'type' to 'checkbox' in 'control_args' array:

```

1  /*
2  * =====
3  * Checkbox field
4  * =====
5  */
6  'new_checkbox_field' => array(
7      'setting_args' => array(
8          'default' => true,
9          'type' => 'option',
10         'capability' => $required_capability,
11         'transport' => 'refresh',
12         'sanitize_callback' => 'thsp_sanitize_cb',
13     ),
14     'control_args' => array(
15         'label' => __( 'New radio control label', 'my_theme_textdomain' ),
16         'type' => 'checkbox', // Checkbox field control
17         'priority' => 2
18     )
19 ),
20

```

Radio and select controls are almost the same, you just need to specify given choices:

```

1  /*
2  * =====
3  * =====
4  * Radio field
5  * =====
6  * =====
7  */
8  'new_radio_field' => array(
9      'setting_args' => array(
10         'default' => 'option-2',
11         'type' => 'option',
12         'capability' => $thsp_cbp_capability,
13         'transport' => 'refresh',
14     ),
15     'control_args' => array(
16         'label' => __( 'New radio control label', 'my_theme_textdomain' ),
17         'type' => 'radio', // Radio control
18         'choices' => array(
19             'option-1' => array(
20                 'label' => __( 'Option 1', 'my_theme_textdomain' )
21             ),
22             'option-2' => array(
23                 'label' => __( 'Option 2', 'my_theme_textdomain' )
24             ),
25             'option-3' => array(
26                 'label' => __( 'Option 3', 'my_theme_textdomain' )
27             )
28         ),
29         'priority' => 3
30     )
31 ),
32
33 /*
34 * =====
35 * =====
36 * Select field
37 * =====
38 * =====
39 */
40 'new_select_field' => array(
41     'setting_args' => array(
42         'default' => 'option-3',
43         'type' => 'option',
44         'capability' => $thsp_cbp_capability,
45         'transport' => 'refresh',

```

```

46     ),
47     'control_args' => array(
48         'label' => __( 'New select field label', 'my_theme_textdomain' )
49         'type' => 'select', // Select control
50         'choices' => array(
51             'option-1' => array(
52                 'label' => __( 'Option 1', 'my_theme_textdomain' )
53             ),
54             'option-2' => array(
55                 'label' => __( 'Option 2', 'my_theme_textdomain' )
56             ),
57             'option-3' => array(
58                 'label' => __( 'Option 3', 'my_theme_textdomain' )
59             )
60         ),
61         'priority' => 4
62     )
63 )

```

And finally, two complex control types that are built into WordPress — file upload and image upload:

```

1  /*
2  * =====
3  * =====
4  * File Upload
5  * =====
6  * =====
7  */
8  'new_file_upload_field' => array(
9      'setting_args' => array(
10         'default' => '',
11         'type' => 'option',
12         'capability' => $thsp_cbp_capability,
13         'transport' => 'refresh',
14     ),
15     'control_args' => array(
16         'label' => __( 'File upload', 'my_theme_textdomain' ),
17         'type' => 'upload', // File upload field control
18         'priority' => 5
19     )
20 ),
21
22 /*
23 * =====
24 * =====
25 * Image Upload

```

```

26 * =====
27 * =====
28 */
29 'new_image_upload_field' => array(
30     'setting_args' => array(
31         'default' => '',
32         'type' => 'option',
33         'capability' => $thsp_cbp_capability,
34         'transport' => 'refresh',
35     ),
36     'control_args' => array(
37         'label' => __( 'Image upload', 'my_theme_textdomain' ),
38         'type' => 'image', // Image upload field control
39         'priority' => 6
40     )
41 )

```

Note that I used **'type' => 'option'** in setting arguments for all of these fields. This will allow all the values to be stored as one value in your database. The alternative is **'type' => 'theme_mod'** but for now let's stick with 'option'.

Using Options Array to Add Customizer Sections, Settings and Controls

If you're not sure [how to interact with WordPress Theme Customizer](#), go and check, because that's what we'll be doing now. The only difference is that instead of adding sections, settings and controls one at a time, we'll automate the process using serialized array we created. Let's just jump into it:

```

1  function thsp_cbp_customize_register( $wp_customize ) {
2
3      /**
4       * Custom controls
5       */
6      require( dirname(__FILE__) . '/custom-controls.php' );
7
8
9      /*
10     * Get all the fields using a helper function
11     */
12     $thsp_sections = thsp_cbp_get_fields();
13
14
15     /*

```

```

16     * Get name of DB entry under which options will be stored
17     */
18     $thsp_cbp_option = thsp_cbp_option();
19
20
21     /**
22     * Loop through the array and add Customizer sections
23     */
24     foreach( $thsp_sections as $thsp_section_key => $thsp_section_value ) {
25
26         /**
27         * Adds Customizer section, if needed
28         */
29         if( ! $thsp_section_value['existing_section'] ) {
30
31             $thsp_section_args = $thsp_section_value['args'];
32
33             // Add section
34             $wp_customize->add_section(
35                 $thsp_section_key,
36                 $thsp_section_args
37             );
38
39         } // end if
40
41         /*
42         * Loop through 'fields' array in each section
43         * and add settings and controls
44         */
45         $thsp_section_fields = $thsp_section_value['fields'];
46         foreach( $thsp_section_fields as $thsp_field_key => $thsp_field_value ) {
47
48             /*
49             * Check if 'option' or 'theme_mod' is used to store option
50             *
51             * If nothing is set, $wp_customize->add_setting method will
52             * If 'option' is used as setting type its value will be stored in
53             * {prefix}_options table. Option name is defined by thsp_cbp_option
54             */
55             if ( isset( $thsp_field_value['setting_args']['type'] ) && 'option' === $thsp_field_value['setting_args']['type'] ) {
56                 $setting_control_id = $thsp_cbp_option . '[' . $thsp_field_key . ']';
57             } else {
58                 $setting_control_id = $thsp_field_key;
59             }
60
61             /*
62             * Add default callback function, if none is defined
63             */
64             if ( ! isset( $thsp_field_value['setting_args']['sanitize_callback'] ) ) {

```

```
65     $thsp_field_value['setting_args']['sanitize_cb'] = 'thsp.  
66 }  
67  
68 /**  
69  * Adds Customizer settings  
70  */  
71 $wp_customize->add_setting(  
72     $setting_control_id,  
73     $thsp_field_value['setting_args']  
74 );  
75  
76 /**  
77  * Adds Customizer control  
78  *  
79  * 'section' value must be added to 'control_args' array  
80  * so control can get added to current section  
81  */  
82 $thsp_field_value['control_args']['section'] = $thsp_section.  
83  
84 /*  
85  * $wp_customize->add_control method requires 'choices' to be  
86  */  
87 if ( isset( $thsp_field_value['control_args']['choices'] ) )  
88     $thsp_cbp_choices = array();  
89     foreach( $thsp_field_value['control_args']['choices'] as  
90         $thsp_cbp_choices[$thsp_cbp_choice_key] = $thsp_cbp_  
91     }  
92     $thsp_field_value['control_args']['choices'] = $thsp_cbp.  
93 }  
94  
95  
96 // Check control type  
97 if ( 'color' == $thsp_field_value['control_args']['type'] )  
98     $wp_customize->add_control(  
99         new WP_Customize_Color_Control(  
100             $wp_customize,  
101             $setting_control_id,  
102             $thsp_field_value['control_args']  
103         )  
104     );  
105 } elseif ( 'image' == $thsp_field_value['control_args']['typ  
106     $wp_customize->add_control(  
107         new WP_Customize_Image_Control(  
108             $wp_customize,  
109             $setting_control_id,  
110             $thsp_field_value['control_args']  
111         )  
112     );  
113 } elseif ( 'upload' == $thsp_field_value['control_args']['ty
```

```

114         $wp_customize->add_control(
115             new WP_Customize_Upload_Control(
116                 $wp_customize,
117                 $setting_control_id,
118                 $thsp_field_value['control_args']
119             )
120         );
121     } else {
122         $wp_customize->add_control(
123             $setting_control_id,
124             $thsp_field_value['control_args']
125         );
126     }
127
128     } // end foreach
129
130 } // end foreach
131
132 }
133 add_action( 'customize_register', 'thsp_cbp_customize_register' );

```

Going through all the sections, adding the ones that don't already exist, then going through all the fields in each section, adding a setting and a control for each. That's all that's going on here.

Remember that we used 'type' => 'option' for all the setting? That's why we now have **"my_theme_options[\$thsp_field_key]"** for both settings and sections. This will store all values as one serialized array that you can retrieve by using **get_option('my_theme_options')**. Or you can just use helper functions defined in **helpers.php** to retrieve both current values and default values for all fields:

```

1  /**
2   * Get Option Values
3   *
4   * Array that holds all of the options values
5   * Option's default value is used if user hasn't specified a value
6   *
7   * @uses   thsp_get_theme_customizer_defaults()   defined in /customize
8   * @return array                                  Current values for a
9   * @since  Theme_Customizer_Boilerplate 1.0
10  */
11  function thsp_cbp_get_options_values() {
12
13      // Get the option defaults
14      $option_defaults = thsp_cbp_get_options_defaults();

```

```

15
16 // Parse the stored options with the defaults
17 $thsp_cbp_options = wp_parse_args( get_option( thsp_cbp_option(), array() ), array() );
18
19 // Return the parsed array
20 return $thsp_cbp_options;
21
22 }
23
24
25 /**
26  * Get Option Defaults
27  *
28  * Returns an array that holds default values for all options
29  *
30  * @uses thsp_get_theme_customizer_fields() defined in /customizer/options.php
31  * @return array $thsp_option_defaults Default values for all options
32  * @since Theme_Customizer_Boilerplate 1.0
33  */
34 function thsp_cbp_get_options_defaults() {
35
36     // Get the array that holds all theme option fields
37     $thsp_sections = thsp_cbp_get_fields();
38
39     // Initialize the array to hold the default values for all theme options
40     $thsp_option_defaults = array();
41
42     // Loop through the option parameters array
43     foreach ( $thsp_sections as $thsp_section ) {
44
45         $thsp_section_fields = $thsp_section['fields'];
46
47         foreach ( $thsp_section_fields as $thsp_field_key => $thsp_field_value ) {
48
49             // Add an associative array key to the defaults array for each field
50             if( isset( $thsp_field_value['setting_args']['default'] ) ) {
51                 $thsp_option_defaults[$thsp_field_key] = $thsp_field_value['setting_args']['default'];
52             } else {
53                 $thsp_option_defaults[$thsp_field_key] = false;
54             }
55
56         }
57
58     }
59
60     // Return the defaults array
61     return $thsp_option_defaults;
62
63 }

```

There's only one more thing I should mention — sanitization callback function that we specified in `setting_args` array. Function is defined in `extend.php` and simply runs data through `wp_kses_post` function:

```
1  /**
2   * Theme Customizer sanitization callback function
3   */
4  function thsp_sanitize_cb( $input ) {
5
6      return wp_kses_post( $input );
7
8  }
```

Where to From Here?

For now, you can use this Theme Customizer Boilerplate in your themes, all you need to do is download it from Github, copy into your theme's directory and include the main file from `functions.php`, which is 100% functional and good enough for most themes.

Since your theme is not "like most themes", next week we'll see how to extend the boilerplate by using its filter and action hooks.

I would love to hear how you think this boilerplate could be improved or expanded, so please fire away in the comments.



Article by [Slobodan Manic](#)

WPExplorer.com author