Getting Started With Custom JavaScript

Getting Started With Custom Code · https://wppopupmaker.com/docs/getting-started-with-custom-code/getting-started-with-custom-js/

Popup Maker relies on JavaScript (JS) to do all of its magic. Without JavaScript, web pages would be downright *boring* (not to mention a lot less functional). Things we take for granted, like popups, alerts, and forms, wouldn't work.

WordPress does a great job of supporting custom JavaScript. But you need to follow the "WordPress Way."

Function Priority (500 for Popup Maker JS)

Pay close attention to the PHP function's priority that loads your JavaScript. If your JavaScript needs to interact with Popup Maker popups, **set the priority to 500 or higher**.

See the code sample below.

So, you want to write some custom JavaScript to customize Popup Maker? This doc outlines best practices and gives you example code to help you get started.

How Do I Add My JavaScript?

We'll cover these 3 ways to add your code:

1. Using a plugin (generally the easiest way)
2. Editing your *child* theme's `functions.php` file
3. Using your theme's built-in settings

Using a Plugin

If you haven't set up a *child* theme or you aren't comfortable editing your `functions.php` file, then using a plugin is the way to go! Now, if you've never heard of a *child* theme or a `functions.php` file, then using a plugin is a no-brainer. We'll cover one plugin we like to use.

Code Snippets Plugin

We like to use the Code Snippets plugin.

After you install and activate the plugin, add your CSS by going to **Snippets** > **Add New** under your wp-admin side menu.

Use the editor to write or paste in your code. You need to "wrap" your JavaScript in PHP (hook) and set the function **priority to 500** to work with Popup Maker popups.

Here's an example of jQuery code that opens a popup.



Lear more about using PHP in our Getting Started With Custom PHP guide.

Create Your Own Plugin

What's that you say? You want to write your own plugin. Knock yourself out! Head over to Pluginception WordPress plugin to get started creating your very own plugin.

Editing Your Child Theme's functions.php File

If you don't want to use a plugin for your custom PHP, you'll need a child theme. After your *child* theme's set up, you can add your functions and hooks directly into the child theme's `functions.php` file.

Note: We use the `wp_footer hook` throughout Popup Maker's documentation. And, If you've *never* edited your child theme's `functions.php` file, please check out our Getting Started with Custom PHP help guide.

Using Your Theme's Built-in Settings

Some WordPress themes have built-in support for adding custom JavaScript. So, check with your theme's documentation or support team first.

Below are a couple of popular themes with links to their how-to guides.

- **Divi**: https://www.elegantthemes.com/blog/divi-resources/best-practices-for-using-external-javascript-snippets-with-divi
- **Thrive**: https://help.thrivethemes.com/en/articles/4430084-how-to-add-scripts-to-a-website-using-thrive-themes

Example PHP Code That Loads Custom JavaScript

Below is a *template* you can use to help you get started. You'll need to add the snippet (except for line #1) using 1 of the 3 ways we covered. Then, replace the popup ID number and change the function name to what you want (function names must be unique).

```
<?php // You can generally ignore this first line when copying to your child theme's functions.php file or to a code snippets plugin.
```

```
/**
* Add custom JavaScript scripts to the footer of your site theme.
*
* @since 1.0.0
*
* @return void
*/
function my_custom_popup_scripts() { ?>
   <script type="text/javascript">
       jQuery(document).ready(function ($) {

           // ADD CUSTOM CODE HERE

           // Example of opening a click trigger popup
           // using the default trigger class. Change 123
           // to your popup ID.
           $(".popmake-123").click(function () {
               PUM.open(123); // Launch popup 123.
           }); // Click

       }); // jQuery
    </script>
<?php }
add_action( 'wp_footer', 'my_custom_popup_scripts', 500 );

/**
* Add to your child theme's functions.php file or use a code snippets plugin.
*/
```

What's happening in that code sample above?

Here's a breakdown of what that code does:

- First, it defines a PHP function called `my_custom_popup_scripts`.
- The `my_custom_popup_scripts` function Injects jQuery code that will open popup 123 when someone clicks an HTML element that has the popmake-123 class, e.g., `<div class="popmake-123">Click Me</div>`.
- Finally, it adds `my_custom_popup_scripts` to the `wp_footer` hook at priority 500 so WordPress inserts the jQuery code to the post's or page's footer after the popups load.

Custom JavaScript that interacts with popups needs to load after popup HTML code on a post or page. A priority of 500+ should be enough to load your JavaScript after the popups load.

Pro Tip

That example calls `PUM.open()` to launch a popup.

See Popup Maker's JavaScript API to learn more.

Basic Troubleshooting

Disclaimer About Using Custom Code Samples

Major caveat: Every time you use custom code, you take responsibility for testing and maintaining it. Here are some best practices you should follow:

- **Back up your site**: Make sure you have a recent full backup of your site.
- **Use a staging site**: Test your code on a staging copy of your live site.
- **Tweak the example**: Remember that any custom code example you decide to reuse is exactly that—an example! Meaning, that you'll need to tweak and make sure the code works in your environment and meets your needs.
- **Disable custom code when troubleshooting conflicts**: Whenever you need to troubleshoot your site for a possible plugin or theme conflict, make sure you disable all custom code.

If your site shows JavaScript or PHP error messages or even a blank white page after making your code changes, you'll need to "back out" or remove your last edits. Here are 4 ways you can do that:

- If you're using **Code Snippets**, deactivate your snippet. For **Simple Custom CSS and JS**, unpublish your code.
- Comment out your edits using `//` for a single line or `/* */` for multiple lines. Save your changes.
- Revert to your backup `functions.php`.
- Restore your site from your last full backup.

Once you've backed out your last edits and your site working again like it was before, you can *debug* the code that caused the error.