Royle Media ▪ 3880 SE 8th Ave, Suite 250 - Portland, OR 97202 ▪ (503) 577-6905

How to Customize the Default Search Form in WordPress

A client whose site we recently rebuilt on WordPress came up with an unusual request: they asked for two search forms on each page--one searching the site, the other a catalog on another site.

Fortunately, the Avada theme we used allows you to have a search form in both the header and the footer, so taking care of that part of the request was easy. It looked to me like the tricky part would be in getting the forms to submit to different locations.

Actually, there was another tricky part before that one: distinguishing between the two forms. The default search form has few distinguishing marks, often only the CSS class “searchform”.

Avada allowed us two search forms but no way in WordPress to tell them apart. At first, I thought I’d end up deep in PHP code, writing my own forms to replace the default searchform.php. This turned out to be not only a dead-end (for so many reasons), but a dead-end that got hijacked in the middle of the project.

The version of Avada we started out with had its own search form code buried deep down in the theme directory structure, which took a support ticket or two to find. About the same time I realized I was never going to hack a clear path through a jungle of conditional code, Avada released an update that abandoned the proprietary search form code in favor of good old searchform.php. After an unsuccessful attempt to wad up my computer monitor as if it were a piece of paper with really bad news, I decided to try a different approach: jQuery.

If you’ve been locked in a bunker since before 2006, jQuery is a cross-platform Javascript library that makes hopping about in the DOM and tweaking on-page elements about 1,000% easier than in native Javascript. The goal was to customize one and only one of the search forms. The code I spent weeks writing to reach this goal from the server side (and which never did) was about as easy to follow as a single strand of spaghetti through an Olive Garden all-you-can-eat lunch. In comparison, the jQuery script I wrote fits in one screen and took a few hours to write.

This article assumes you have access to the file and directory structure of your WordPress installation, so if you can’t FTP in, this probably isn’t for you. And before I show you how the script works, you need to understand that creating a form that submits data to another site without the site-owner’s knowledge and permission amounts to a cross-site scripting attack. You're using your form to inject data into the other site's code.

In the case of the catalog site, the vendor allows customers to create their own off-site search forms and even advises them on how to do it. Obviously, they've taken extra measures to sanitize the posted data, and obviously your form needs extra measures to sanitize the data before it's posted. (For this article, I cover only the basics of jQuery, not the extra measures.) Unless you know you have the blessing of the other site, then you're attacking the other site, whatever your intentions.

That being said, here's how you can customize the default WordPress search form action with jQuery. That is, after you've also used jQuery to find one search form among its clones.

WordPress has included the jQuery library for several versions now, so you don't have to worry about including the library with script tags. However, there is a right way to include your script in WordPress, which we'll look at first.

While it's tempting to treat WordPress like static HTML and simply add your jQuery script to the <head> section of the site with script tags in your theme's header.php, that's not the way to do it. For one thing, your script will be overwritten the next time the theme is updated. In WordPress, it's best to enqueue the script, in WordPress-speak, in another theme file called functions.php. You'll still lose your work the next time the theme is updated, so to enqueue your scripts correctly and have them survive updates, create a child theme. (If you're still not using child themes, you really should be. Here's the guide.)

For our needs, I created a blank search-form.js file and saved it to a "js" directory I added to the directory of child theme we made for Avada.

I then downloaded a copy of the child theme's functions.php file and added this code from the WordPress codex:

```
// From https://codex.wordpress.org/Function_Reference/wp_enqueue_script

function my_scripts_method() {
// register your script location, dependencies and version
    wp_enqueue_script(
        'search_form',
        get_stylesheet_directory_uri() . '/js/search-form.js',
        array('jquery'),
        '1.0.0',
        true
       );
}

add_action('wp_enqueue_scripts', 'my_scripts_method');
```

I would urge you to read carefully the Codex Function Reference on wp_enqueue_script, but for now, focus on the argument that includes the function get_stylesheet_directory_uri(). Immediately following is the path to my script: /js/search-form.js. Replace that path with the path to your script, keeping in mind that in the world of WordPress, the root path "/" is not the WordPress directory itself, but rather the active theme directory *within* the wp-content directory *within* the WordPress directory. In this case, "/" is the Avada child theme directory.

It may seem odd to enqueue the script before writing it, but it's helpful first to test to see if your script is enqueued properly. Adding a simple "Hello World" Javascript alert box to search-form.js will quickly tell you what you need to know. Once you know WordPress is managing your script, you can start adding jQuery code.

A web page can't be manipulated by jQuery until it's "ready"—that is, until the page has completely loaded in the browser. So everything in jQuery happens inside the function **$(document).ready()**, which is always the first line of every jQuery script:

```
$(document).ready()
```

That "$" is shorthand for "jQuery" and much quicker to type, but your theme might be using other Javascript libraries which also like to use the "$," so it's best to put jQuery into no-conflict mode. If you want to keep using "$" for "jQuery," you could add a line before the **(document).ready()** function:

```
var $j = jQuery.noConflict();
```

Then you could write

```
$j(document).ready()
```

Since I didn't care whether I used the "$," I just used term "jQuery" itself:

```
jQuery(document).ready()
```

Starting out, the full code block for search-form.js looked like this:

```
jQuery(document).ready(function() {
    //All my code here;
});
```

First, I needed to find the first search form on the page. Remember that the default WordPress search form is marked up with the CSS class “searchform.” If there were only one search form on the page, you’d find it with this statement:

```
jQuery(document).ready(function() {
    jQuery(".searchform");
});
```

Mind you, that statement isn’t doing anything yet but finding the one search form on the page. We’ll do stuff to the search form in a bit, but remember, there’s more than one search form. Fortunately, jQuery offers you all kinds of pseudo-classes to make element selection easy. Since it happens that the search form I wanted to customize was the first one to appear on the page, I took advantage of the “:first” pseudo-class right away:

```
jQuery(document).ready(function() {
    jQuery(".searchform:first");
});
```

Now it doesn’t matter how many other search forms there are on the page. I’ve found the first one, the one I need. To change where the form submits to, you have to change the action attribute of the form. The form action of the default search form is a PHP variable means “this site”:

```
<form class="searchform" action="<?php echo esc_url( home_url( '/' ) ); ?>">
```

In my jQuery script, I’m changed that form action to the URL of the online catalog, using the jQuery “.attr()” method, a means for changing a DOM element’s attributes. The “.attr()” method takes two arguments: the name of the attribute you want to change and the new value:

```
jQuery(document).ready(function() {
    jQuery(".searchform:first").attr("action", “http://example.com”); });
});
```

Since the script is already enqueued, you can test your jQuery from within WordPress. If you’re following along, upload the script, refresh the page, and view the source of the page. Find the search form code. The first line should look like this:

```
<form class="searchform" action="http://example.com">
```

After the page has loaded, jQuery finds the first search form and replaces the form action with the desired URL. Note: using Firebug in Firefox or another console tool in your favorite browser will show you any scripting errors you may have caused.

So the form action is changed, but the search form isn't ready to use yet. Trying a search will at best get you a 404 error on the target site. That's because the target site is expecting more than just a search term. It's looking for the search term, metadata about the search term, and the method used to submit the search term.

First, let's give it the search term itself. We need the contents of the search form's text box, but we also need to send it in the way the target site expects it. Thanks to the vendor's API, I knew how the search term needed to be sent. So instead of just finding the search term and sending it, I replaced the search box form element itself, using jQuery's **.replaceWith()** method:

```
jQuery(".s:first").replaceWith('<input type="text" value=""
placeholder="Search catalog" name="term" class="s">');
```

In this example, I've replaced the search form's text input element with the API's recommendation.

Notice **jQuery(".s:first")**. Why not **jQuery(".searchform:first")**? With a little tweaking, you could use that, but WordPress very helpfully marks up the text input element in the search form with a CSS class "s." Using that class as your selector not only makes it easy to find, but adding it to your new text input element (class="s") allows you to keep the theme's styling of the search form.

Here's my evolving search-form.js code:

```
jQuery(document).ready(function() {jQuery(".searchform:first").attr("action",
"http://example.com"); });

    jQuery(".s:first").replaceWith('<input type="text" value=""
placeholder="Search catalog" name="term" class="s">');

});
```

The API adds metadata to the search term, including the client's ID, a language code, and the search type (keyword, title, subject, etc.). It adds them in the form of hidden input elements, none of which exist in the default WordPress search form, so there's nothing to replace. Instead, we'll add them to the search form with jQuery's **.append()** method. Here's the line adding the hidden input indicating the search type:

```
jQuery(".searchform:first").append('<input type="hidden" value="Keyword"
name="type">');
```

Adding that and other hidden elements gives us this searchform.js:

```
jQuery(document).ready(function() {
    jQuery(".searchform:first").attr("action", “http://example.com”); });
    jQuery(".s:first").replaceWith('<input type="text" value=""
placeholder="Search catalog" name="term" class="s">');
    jQuery(".searchform:first").append('<input type="hidden" value="Keyword"
name="type">');
    jQuery(".searchform:first").append('<input type="hidden" value="en"
name="LanguageID">');
    jQuery(".searchform:first").append('<input type="hidden" value="1234"
name="ClientID">');
});
```

Now we’ll add the form submission method the API expects. Form data is submitted either by the GET method, which adds the form data to the URL, or by the POST method, which sends the form data via HTTP. The default WordPress search form uses GET, but the catalog vendor’s API says POST, so we’ll use POST, again by using the jQuery “.attr()” method:

```
jQuery(".searchform:first ").attr("method", "POST");
```

With that line added, searchform.js looks like this:

```
jQuery(document).ready(function() {
    jQuery(".searchform:first").attr("action", “http://example.com”); });
    jQuery(".s:first").replaceWith('<input type="text" value=""
placeholder="Search catalog" name="term" class="s">');
    jQuery(".searchform:first").append('<input type="hidden" value="Keyword"
name="type">');
    jQuery(".searchform:first").append('<input type="hidden" value="en"
name="LanguageID">');
    jQuery(".searchform:first").append('<input type="hidden" value="1234"
name="ClientID">');
    jQuery(".searchform:first ").attr("method", "POST");
});
```

All that’s left to do is submit the form. Here’s the line to add:

```
jQuery(".searchform:first").submit();
```

But if that's all you do, the form won't work. You'll either get a Javascript error or you'll find WordPress trying to do a site search with the search term you supplied, even though you've changed the form data's destination. (This drove me crazy until I figured it out.)

Because of the way form submit buttons work, you have to take control of what happens when you click the button and then submit the form data when you're ready. To do that, I took advantage of WordPress's marking up the submit button with the CSS class "searchsubmit" to select it. Then I used jQuery's **.click()** method to modify the button's "click" event before submitting the form. Here's the basic code block:

```
jQuery(".searchsubmit:first").click(function(){
    // Add code here to modify the submit button's behavior when you click
it.
    jQuery(".searchform:first").submit();
});
```

But what to add? I've modified the search box itself, but I need to grab the contents of the search box, using jQuery's **.val()** method:

```
jQuery(".s:first").val();
```

The submit button's click event is also the right place to add the lines changing the form's action and method, so the function ends up looking like this:

```
jQuery(".searchsubmit:first").click(function(){
    jQuery(".s:first").val();
    jQuery(".searchform:first").attr("action", “http://example.com”); });
    jQuery(".searchform:first ").attr("method", "POST");
    jQuery(".searchform:first").submit();
});
```

The final version of searchform.js looks like this:

```
jQuery(document).ready(function() {
    jQuery(".s:first").replaceWith('<input type="text" value=""
placeholder="Search catalog" name="term" class="s">');
    jQuery(".searchform:first").append('<input type="hidden" value="Keyword"
name="type">');
    jQuery(".searchform:first").append('<input type="hidden" value="en"
name="LanguageID">');
    jQuery(".searchform:first").append('<input type="hidden" value="1234"
name="ClientID">');
    jQuery(".searchsubmit:first").click(function(){
        jQuery(".s:first").val();
    jQuery(".searchform:first").attr("action", "http://example.com"); });
    jQuery(".searchform:first ").attr("method", "POST");
        jQuery(".searchform:first").submit();
    });
});
```

The script has been properly enqueued in the WordPress child theme. After the page is fully loaded, the first default WordPress search form is modified according to the catalog vendor's API. When the submit button is clicked, the search term and all hidden input data is sent to the new destination.

The jQuery library is the Swiss army knife of the Web, and every WordPress developer should have some familiarity with it. You can sometimes solve problems there are no plugins for.

Mark Price
mark@roylemedia.com