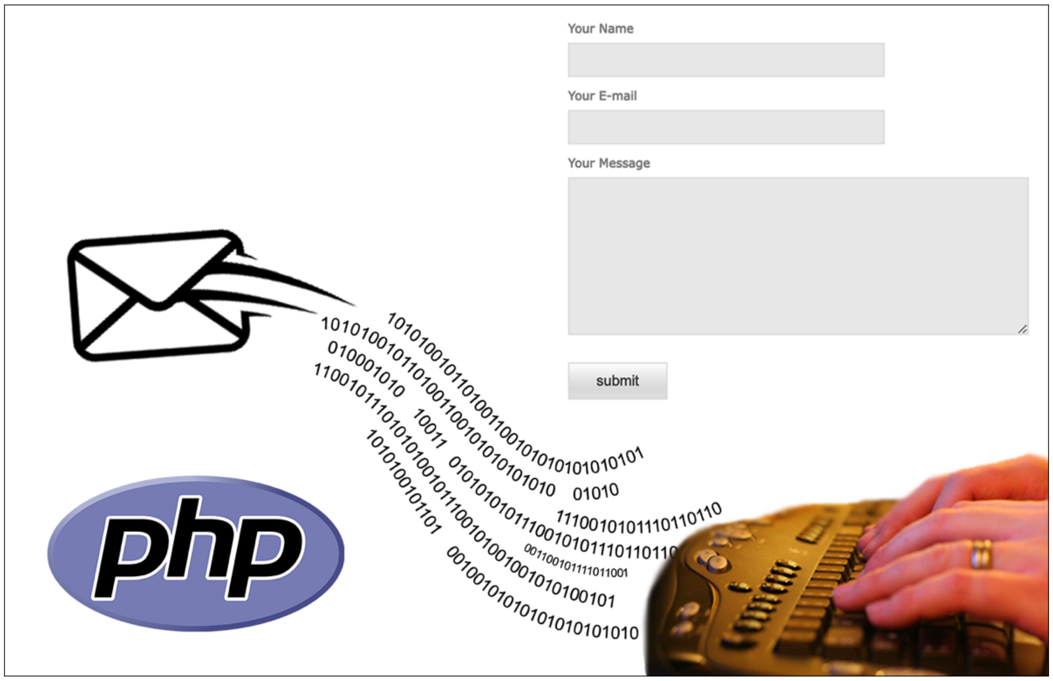
Writing A Contact Form Using PHP

By Chad Jordan – March 24th – 2014



Introduction

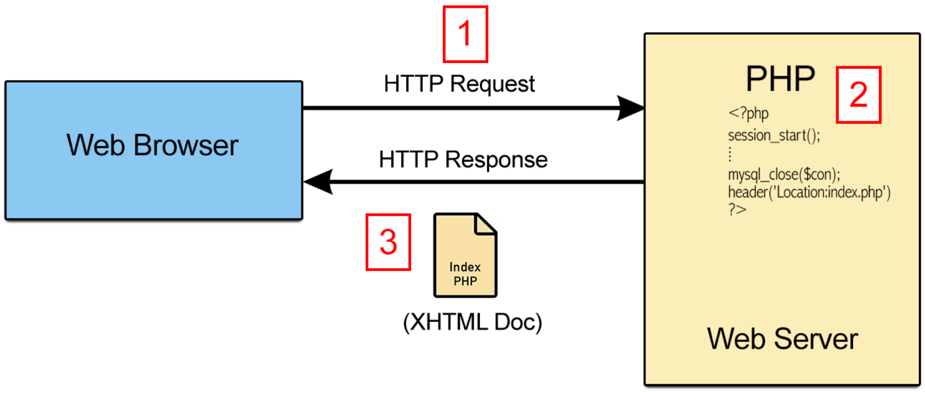
In this guide you will learn:

1. The fundamentals of data communication between HTTP requests and responses
2. Essential markup principles for a clean and simple layout using HTML and CSS
3. A back-end implementation for a simple contact form using PHP

PHP Hypertext Preprocessor is the recursive acronym for the open-source, server-side programming language known as PHP. PHP has evolved very fast in its maturity on the web, climbing swiftly from version to version. With PHP, we can:

1. Encrypt data
2. Generate dynamic page content
3. Create, open, read, write, delete, and close files on the server
4. Add, delete and modify data in a database
5. Collect form data
6. Send and receive cookies
7. Control user-access

Let's briefly look at how PHP communicates. The XHTML *Content-Type* is a text/html response header informing the client that the server has returned HTML. The PHP code automatically generates the HTTP Requests and Responses. An example of making an **HTTP request** to the server and the corresponding **HTTP response** from the server. The PHP processor has two modes of operation, copy mode and interpret mode. It takes a PHP document file as input and produces an XHTML document file. When the PHP processor finds XHTML code *(which may include embedded client-side script)* in the input file, it simply copies it to the output file. When it encounters a PHP script in the input file, it interprets it and sends the output of the script to the output file. PHP is typically interpreted, as is the case with JavaScript. However, PHP implementations perform some precompilation, at least on complex scripts, which increases the speed of interpretation. PHP uses dynamic typing, as does JavaScript. Variables are not type declared, and they have no intrinsic type. The type of a variable is set every time it is assigned a value, taking on the type of that value. Similar to JavaScript, PHP is far more forgiving than most common programming languages. Dynamic typing is largely responsible for this, but the dynamic nature of its strings and arrays also contributes. PHP has an extensive library of functions, making it a flexible and powerful tool for server-side software development. Many predefined functions are used to provide interfaces to other software systems such as mail and database systems. As is the case with JavaScript and Perl, language processors for PHP are free and easily obtainable. Additionally, the PHP processor is an open source system, available on all platforms.

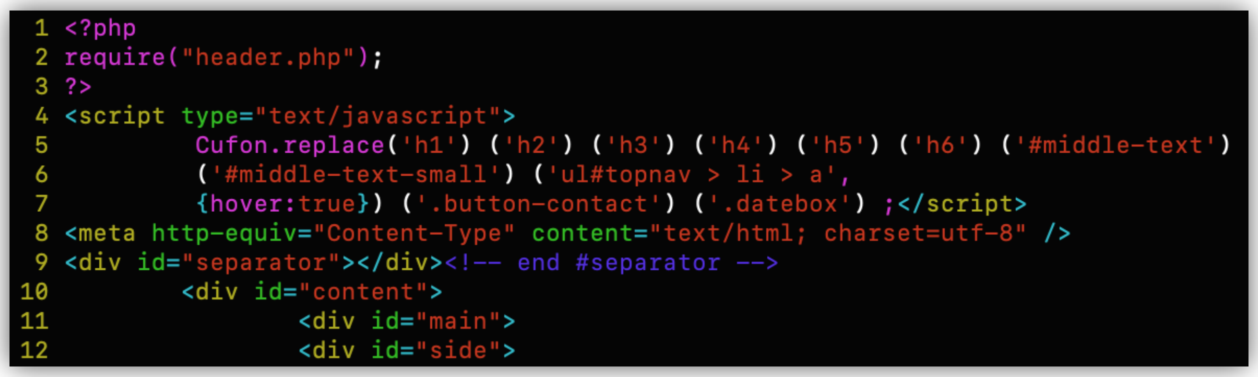


```
<?php
session_start();
⋮
mysql_close($con);
header('Location:index.php')
?>
```

In the past, I have written my fair share of PHP code for contact forms on Flash websites, my PHP contact form on my old website, corporate data encryption, and used PHP to handle various content across all of my websites. Now, I am using PHP to create another simple contact form on my latest website with a few new features and a sleeker layout. The purpose of this guide is to provide a straightforward visual demonstration of creating and embedding your own contact form into a website. I will be using PHP code and other markup code directly from my website, integrated into this guide. My code snippets are demonstrated using the Vim code editor in Linux.

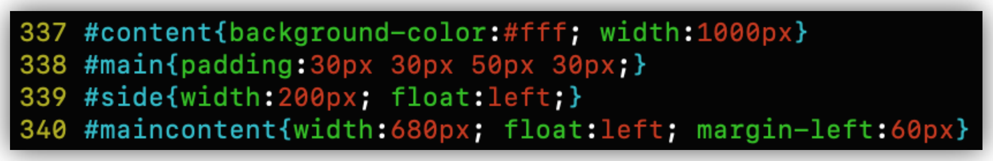
Creating the Form Layout

Most of my PHP code can be transferred from my previous contact form, but the markup portion of the form is what I'm making significant changes to. For my website, I create a file called *contactform.php* which will be used to display the completed form online. Within the file, I begin by using a require function to call all of the code in my header file to be displayed at the top of my page. This saves on loading time, and not having to completely copy and paste all of the extra code every time on each page. On **line 5** I'm using a small CSS trick known as the *Cufon.replace* function because for this page, I don't want the same CSS elements tied to my headers and the middle text of my contact form. Since I'm making a function call, it needs to be placed inside of a **<script>** tag for JavaScript. I then add a div id called *separator* on **line 9** so



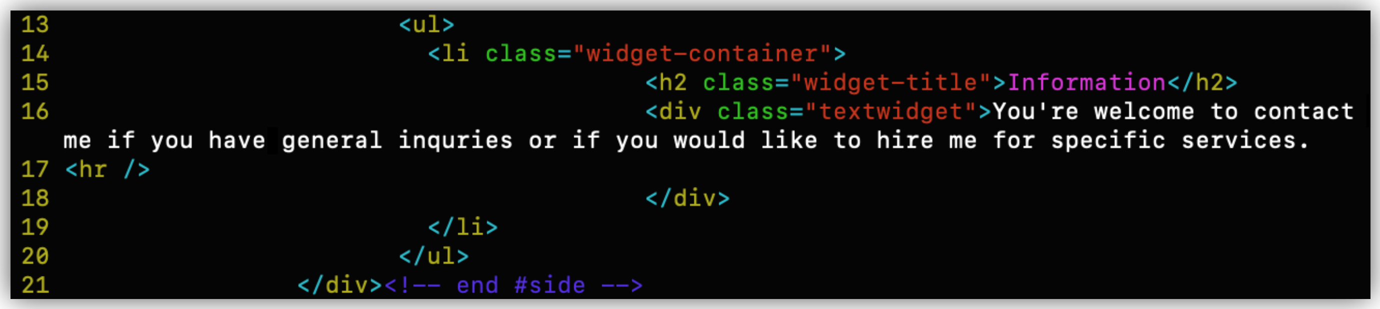
```
 1 <?php
 2 require("header.php");
 3 ?>
 4 <script type="text/javascript">
 5          Cufon.replace('h1') ('h2') ('h3') ('h4') ('h5') ('h6') ('#middle-text')
 6          ('#middle-text-small') ('ul#topnav > li > a',
 7          {hover:true}) ('.button-contact') ('.datebox') ;</script>
 8 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 9 <div id="separator"></div><!-- end #separator -->
10         <div id="content">
11                 <div id="main">
12                 <div id="side">
```

that I can have a versatile, and editable content block for the user should they wish to add custom content. The separator content is entirely up to the user, and it won't be seen until the user decides to add to it. As far as the CSS elements *(div id's)* for my content, the following CSS code is well into my stylesheet, and covers the elements used for my website, but also what is used to wrap around the contact page. In my style.css file, **Line 337** creates the background color, and sets the width of my container. The *main* id is set to four-pixel inputs from left to right at **top, right, bottom,** and **left**. The *side* id is simply setting the width of the element at 200px and float left, *(position left)* on the screen. The *maincontent* id covers the content placed within the main content element.



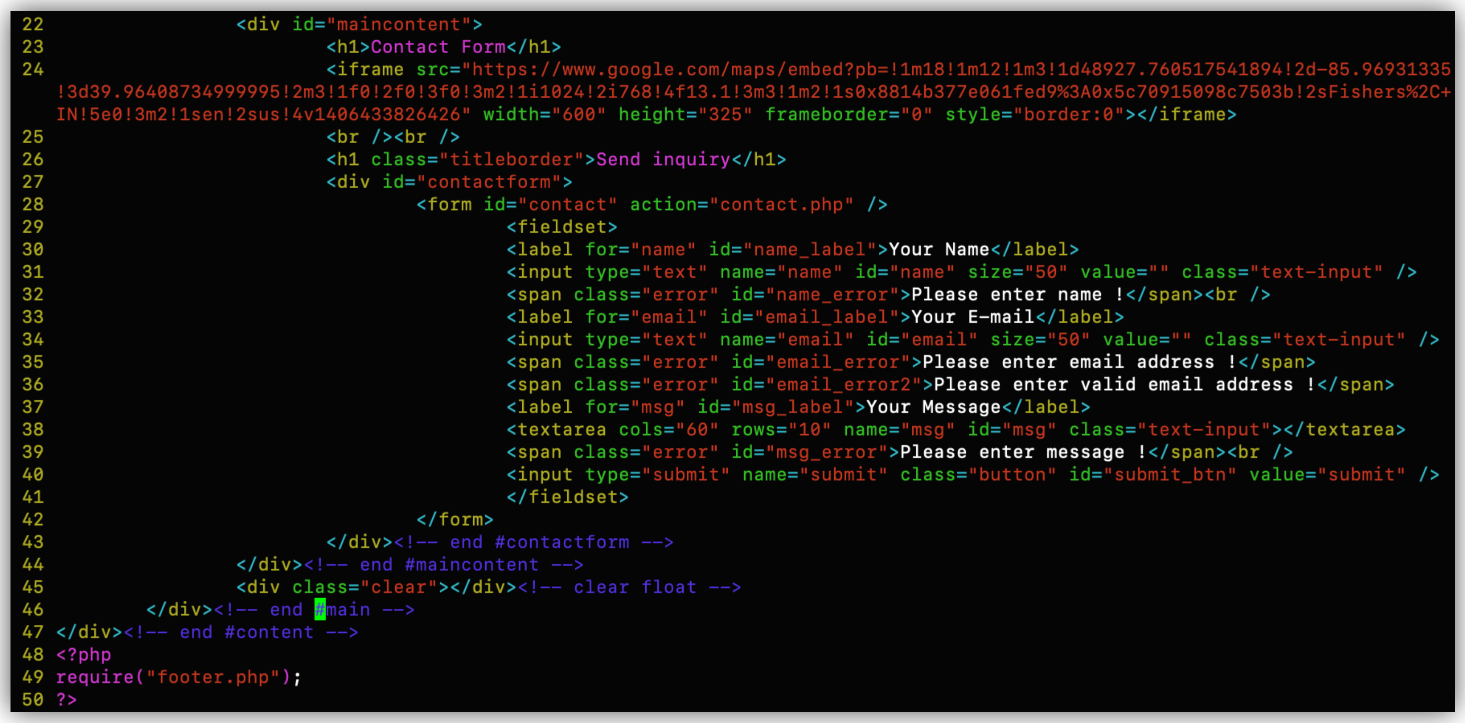
```
337 #content{background-color:#fff; width:1000px}
338 #main{padding:30px 30px 50px 30px;}
339 #side{width:200px; float:left;}
340 #maincontent{width:680px; float:left; margin-left:60px}
```

The widget container is designed to hold any content I wish to add such as website widgets, icons, hyperlinks, or text for information. I place my **list items** inside of an unordered list to contain lists of items *(UL element with LI elements)* to be semantically correct about it. This also helps screen readers and other technologies that depend on correct semantics to work.



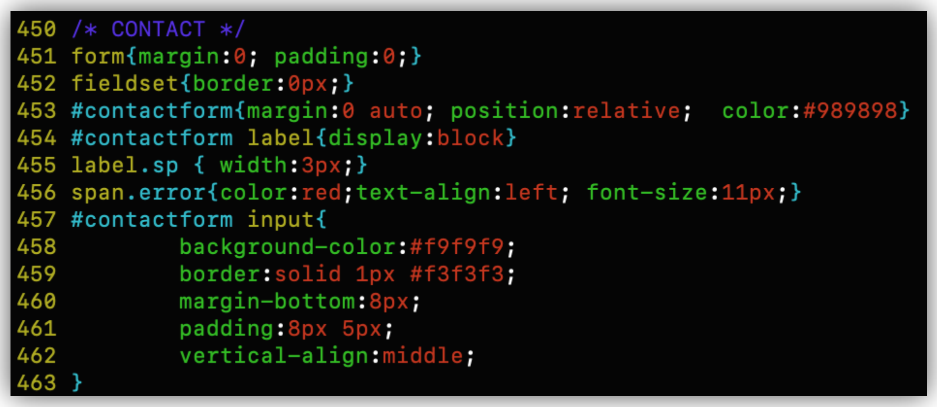
```
13                          <ul>
14                            <li class="widget-container">
15                                           <h2 class="widget-title">Information</h2>
16                                           <div class="textwidget">You're welcome to contact
   me if you have general inquries or if you would like to hire me for specific services.
17 <hr />
18                                           </div>
19                            </li>
20                          </ul>
21                   </div><!-- end #side -->
```

After placing my created content for the *maincontent* id, and the title header, I insert an embedded map of my location from Google Maps. This is obviously not required, but I personally felt it added a nice element to a contact page. The **<iframe>** tag is a commonly used tag for embedding dynamically updated content, and or simply for the sake of an embedded document like this guide. The nice thing about using Google Maps, is the map information will



```
22                <div id="maincontent">
23                        <h1>Contact Form</h1>
24                        <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d48927.760517541894!2d-85.96931335
   !3d39.96408734999995!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x8814b377e061fed9%3A0x5c70915098c7503b!2sFishers%2C+
   IN!5e0!3m2!1sen!2sus!4v1406433826426" width="600" height="325" frameborder="0" style="border:0"></iframe>
25                        <br /><br />
26                        <h1 class="titleborder">Send inquiry</h1>
27                        <div id="contactform">
28                                <form id="contact" action="contact.php" />
29                                        <fieldset>
30                                        <label for="name" id="name_label">Your Name</label>
31                                        <input type="text" name="name" id="name" size="50" value="" class="text-input" />
32                                        <span class="error" id="name_error">Please enter name !</span><br />
33                                        <label for="email" id="email_label">Your E-mail</label>
34                                        <input type="text" name="email" id="email" size="50" value="" class="text-input" />
35                                        <span class="error" id="email_error">Please enter email address !</span>
36                                        <span class="error" id="email_error2">Please enter valid email address !</span>
37                                        <label for="msg" id="msg_label">Your Message</label>
38                                        <textarea cols="60" rows="10" name="msg" id="msg" class="text-input"></textarea>
39                                        <span class="error" id="msg_error">Please enter message !</span><br />
40                                        <input type="submit" name="submit" class="button" id="submit_btn" value="submit" />
41                                        </fieldset>
42                                </form>
43                        </div><!-- end #contactform -->
44                </div><!-- end #maincontent -->
45                <div class="clear"></div><!-- clear float -->
46        </div><!-- end #main -->
47 </div><!-- end #content -->
48 <?php
49 require("footer.php");
50 ?>
```

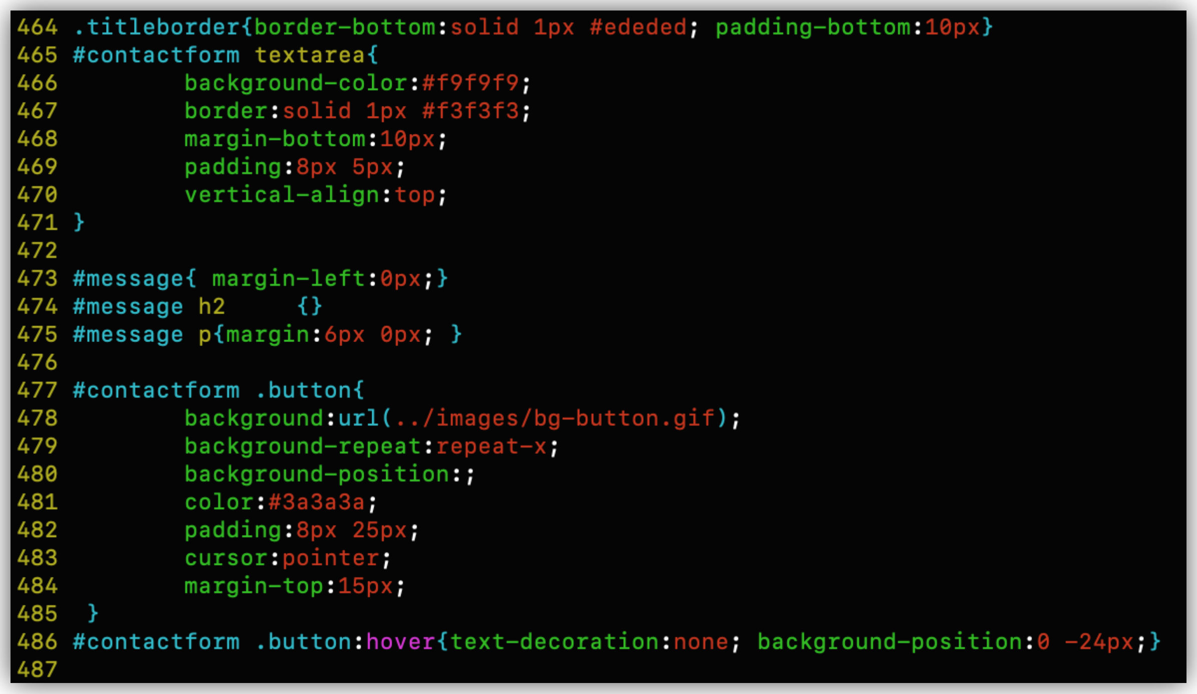
always be dynamically updated over time by Google, which means that the embedded code on **line 24** will never need to be altered unless I decide to change the location of my business. If that is done, I simply grab the updated code from Google Maps and embedded it back inside of the **<iframe>** tag. Next, I place a couple of break tags, create my inquiry text above the form, and then call the div and form id elements for the contact form. The following CSS code is referenced by the *contactform* tag on **line 27:**



```
450 /* CONTACT */
451 form{margin:0; padding:0;}
452 fieldset{border:0px;}
453 #contactform{margin:0 auto; position:relative;  color:#989898}
454 #contactform label{display:block}
455 label.sp { width:3px;}
456 span.error{color:red;text-align:left; font-size:11px;}
457 #contactform input{
458         background-color:#f9f9f9;
459         border:solid 1px #f3f3f3;
460         margin-bottom:8px;
461         padding:8px 5px;
462         vertical-align:middle;
463 }
```

On **line 451** of my stylesheet, I set the element *form* to zero for margin and padding, a very common practice for web development. By eliminating all elements of margin and padding, the user can feel free to go about styling other CSS elements without restraint across all browsers. As I mentioned in my previous guide entitled, *'Coding A Basic Website'* consistency is very important when creating web pages on your websites and I mentioned several of these aspects of CSS in that guide. On **line 452**, the *fieldset* element is used in CSS to group elements within a form. On **line 455**, label.sp *(label spacing)* is a CSS element that controls pixel density for viewing text on android devices. The value of 3px is added to the natural spacing of the characters of the text. Span.error on **line 456** is simply setting the format to the error messages should the form need to print any to the user. The remaining code in the above block is just basic styling for the input of the contact form.

This next block is more of the same styling for the textarea of the contact form. The remaining CSS is mostly just positioning of the elements within the form. The message ids from lines **473** to **475** is controlling the format of the message field and the text within that field.



```
464 .titleborder{border-bottom:solid 1px #ededed; padding-bottom:10px}
465 #contactform textarea{
466         background-color:#f9f9f9;
467         border:solid 1px #f3f3f3;
468         margin-bottom:10px;
469         padding:8px 5px;
470         vertical-align:top;
471 }
472
473 #message{ margin-left:0px;}
474 #message h2     {}
475 #message p{margin:6px 0px; }
476
477 #contactform .button{
478         background:url(../images/bg-button.gif);
479         background-repeat:repeat-x;
480         background-position:;
481         color:#3a3a3a;
482         padding:8px 25px;
483         cursor:pointer;
484         margin-top:15px;
485  }
486 #contactform .button:hover{text-decoration:none; background-position:0 -24px;}
487
```

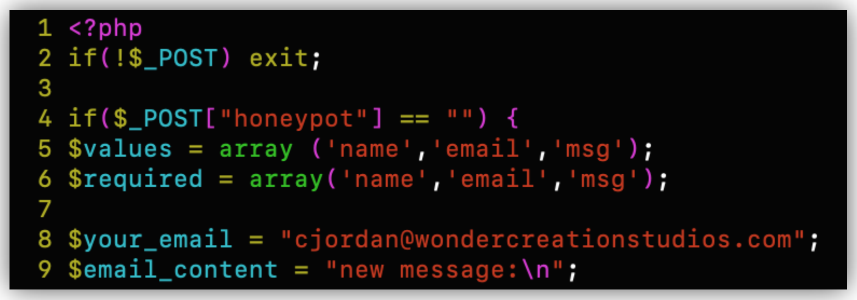
Scrolling back up to the previous XHTML block on **line 28** the form id needs to reference the PHP logic for the form functionality. This is why we set:

```
<form id="contact" action="contact.php" />
```

because *contact.php* will be the file used to link the logic to the contact form. The **<fieldset>** tag on **line 29** begins all user input tags, labels and span tags for the form. The remaining closing tags simply close out the remainder of the form. I've provided HTML comments beside each closing tag to signify which individual portion of the content id that is being closed out.

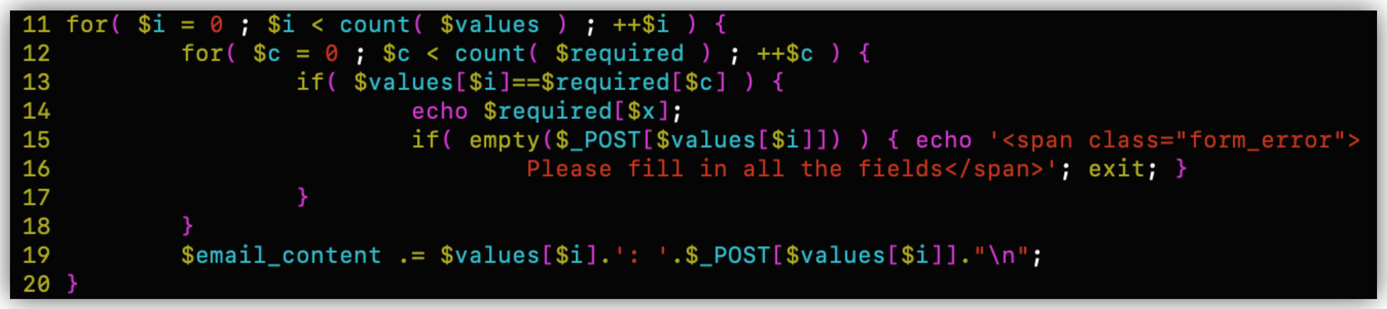
Writing the Form Logic

Writing the logic for the contact form is relatively straightforward. Starting on **line 2**, the ! *(not)* logical operator returns true if the argument on its right-hand side is not true. It forces the argument to be evaluated as a Boolean. In this case, when the **$_POST** array is evaluated as a Boolean, it will be true if it is not empty and false if it is. Next, from **line 4** to **line 7** is how we check for spam. Consider that a lot of spam is checked by using captcha mechanisms. Captchas are also generally inaccessible for people with disabilities. Another method without making the user hit redundant matching images is called **honeypot**. Honeypots are passive, and often implemented as hidden form fields that bots will fill out and humans won't. If that hidden form field has a value, then you know the submission is from a bot and can be rejected. If honeypot is filled *(or hits an error)*, the form is told to investigate the required values specified in the arrays. Next, on **line 8** I specify the email to which the message will be sent and then assign the message string to the *email_content* variable.



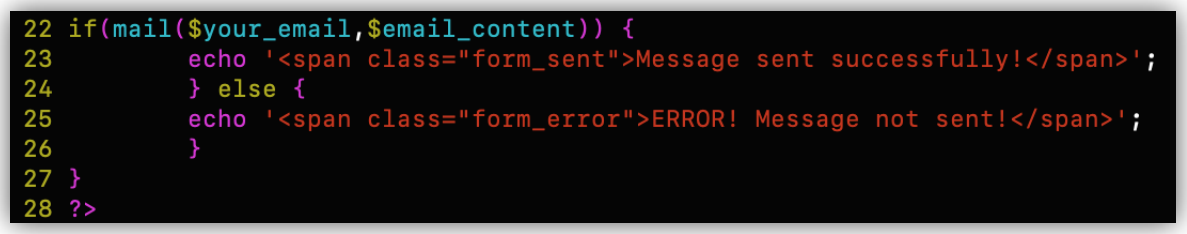
```
1 <?php
2 if(!$_POST) exit;
3
4 if($_POST["honeypot"] == "") {
5 $values = array ('name','email','msg');
6 $required = array('name','email','msg');
7
8 $your_email = "cjordan@wondercreationstudios.com";
9 $email_content = "new message:\n";
```

The for loop in this next block checks for the required values in the fields, and if it finds an error, it prints off a message to fill in all of the fields. **Line 19** takes the values in the associative arrays, collects from the form data with **$_POST** and then stores those values back into email_content which makes up the string data for *"new message"*



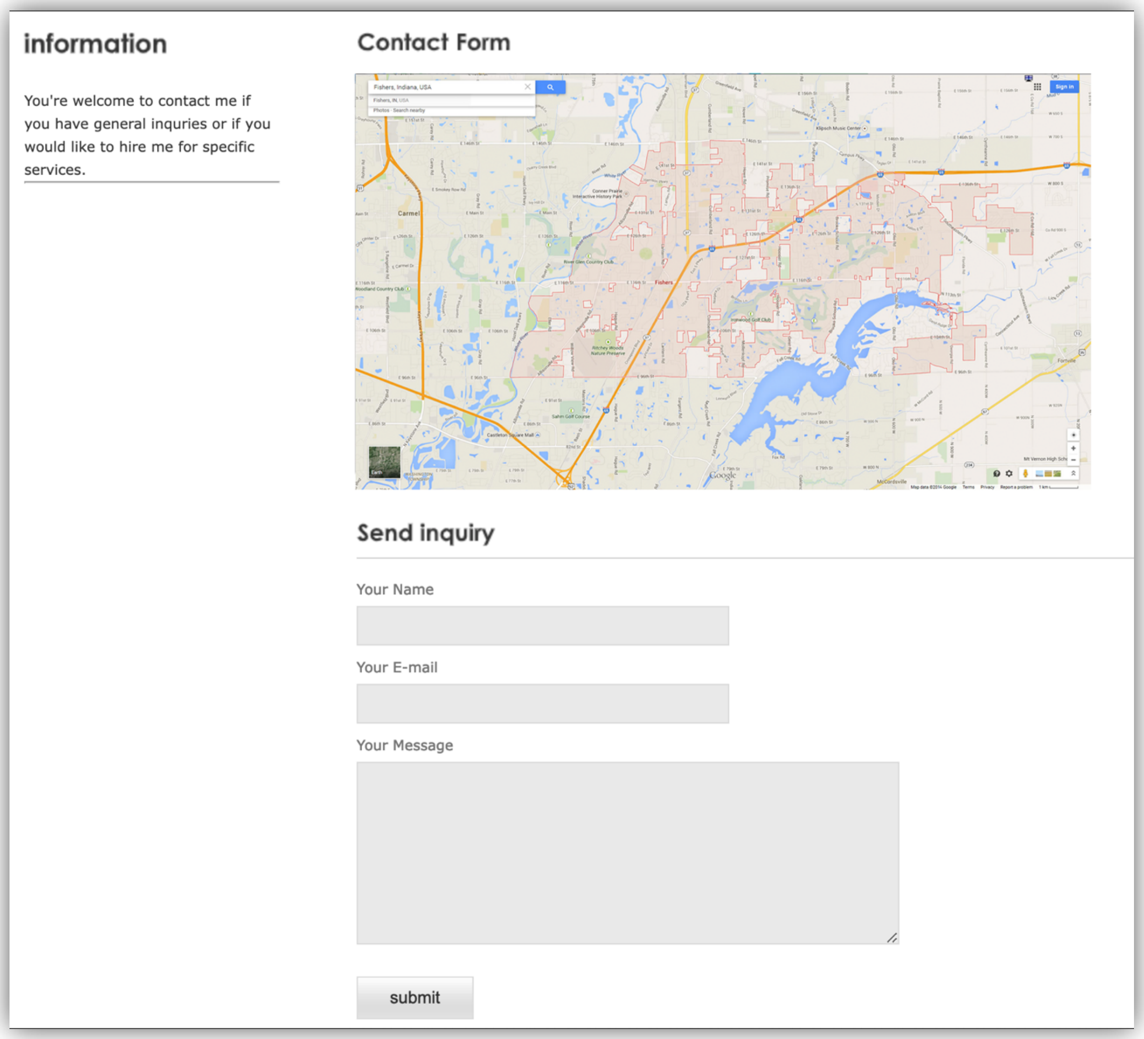
```
11 for( $i = 0 ; $i < count( $values ) ; ++$i ) {
12         for( $c = 0 ; $c < count( $required ) ; ++$c ) {
13                 if( $values[$i]==$required[$c] ) {
14                         echo $required[$x];
15                         if( empty($_POST[$values[$i]]) ) { echo '<span class="form_error">
16                                 Please fill in all the fields</span>'; exit; }
17                 }
18         }
19         $email_content .= $values[$i].': '.$_POST[$values[$i]]."\n";
20 }
```

If all statements pass, then we print a message that the message was sent successfully, otherwise if not, the user receives an error that the message was not sent.



```
if(mail($your_email,$email_content)) {
        echo '<span class="form_sent">Message sent successfully!</span>';
        } else {
        echo '<span class="form_error">ERROR! Message not sent!</span>';
        }
}
?>
```

Once I send the files for the contact form to my server, this is the following result on my website:



information

You're welcome to contact me if you have general inquries or if you would like to hire me for specific services.

Contact Form

Send inquiry

Your Name

Your E-mail

Your Message

That's it! This contact form has been tested and is fully functional for any website. Want to test it? Click here
While I'm sure my contact page will change again over time, I will make sure the same link is always active for any potential readers that may wish to test it.

Conclusion

My hope is that this guide has been helpful in teaching the process of how contact pages interact and function using PHP programming. I've more recently started learning Laravel PHP, and would love to put a guide together about the advantages of using frameworks in PHP. Implementing authentication logic is a good advantage of using Laravel and its security advantages could make for a great topic. All diagrams and code presented in this guide were created and written by Chad Jordan for learning purposes only. This PHP code is currently from the latest version 5.5.10, the above screenshot of the contact form was displayed on Chrome using Ubuntu Linux 14.04.6 and written using the Vim code editor. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com or contact me using my latest contact form!

Resources Used:

- McFarland, Sawyer David – *CSS: The Missing Manual – 1st Edition* – 2006
- W3schools.com
- PHP.net