(For the latest PDF files Merger Software please have a look at the top of the left margin, inside the red box.)

Continues From Part 7

We finished the entire code we needed in Part (7). Now we start to understand each line. My first task is to keep the values entered into all fields as they are. That is possible by keeping "Forms *inputs*" of ***HTML*** parts valid and alive using ***PHP*** built-in functions.

First note that the form tag action is empty and won't dispatch any action to a server-side scripting page.

```
101 <form method="post" action="">
```

Next, please look at line (**103**) to see how ***PHP*** puts value in an ***HTML*** tag.

```
103 <input type="text" name="Name" id="Name" value="<?php echo
isset($_POST['Name']) ? $_POST['Name'] : '' ?>" />
```

"*echo*" is the PHP print command. Then, you can see the C language *if-else* style, common in many programming languages. You can put *if-else* if you prefer the longer style. Hence,

```
1 if (some_thing){
2                   do_some_job;
3               }
4               else{
5                   do_else;
6               }
```

can be written in shorter form as,

```
some_thing ? do_some_job : do_else;
```

PHP, used at that position, won't need the finishing semi-colon. Therefore, if the field is empty, e.g., when you first open the page, keeps it empty; otherwise, if it has some value keeps them as they are. Also there are warning and cautions accessing super-global such as ***$_POST['Name']*** in carefully written ***PHP*** files. They need filtering functions such as using ***filter_input (INPUT_POST, 'Name')*** . I avoided those details for enthusiasts in learning ***PHP***, as I found them beyond a simple contact form. You also might like to guard fields of your form with ***PHP*** function ***htmlentities ( )***.

Please note the difference of *textarea* tag. *textarea* is a legacy of ancient computing (I mean 1990's) and has its own specification. It accepts *carriage-return*, *new-line*, *carriage-return and new-line*, *new paragraph*, *tab*, *Home*, *End*, *Page-Down*, and *Page-Up* as it *scrolls*. These are not settled issues adding internationalisation problems to them ever increasingly. Anyway difference is not much. Value cannot be *attributed* inside the tag, similar to other inputs. We put ***PHP*** code outside the tag.

```
115  <textarea name="Message" rows="20" cols="20" id="Message"><?php echo
isset($_POST['Message']) ? $_POST['Message'] : '' ?></textarea>
```

Now bring your server-side *demoPoster.php* file into your ***HTML*** contact form. Remember that you have saved your contact form as a ***PHP*** file with .php extension. Here I have called it *demoFinal.php*.

You may notice some changes from previous *demoPoster.php*. All the code becomes active on the condition that you "*submit*" the form. Otherwise when a user opens the page for the first time it will dispatch errors regarding empty fields and wrong reCaptcha. This is line **(130),**

```
130    if( isset($_POST['submit']) ) {
```

and ends at line **(223).**

In the next line **(131)** we have identified and initialised a place holder for errors of the visitor.

```
130    $error = null;
```

Call to *recaptchlib.php* library and introducing the "private key" is done in next two lines (**132**) and (**133**).

```
132     require_once('recaptchalib.php');
133     $privatekey = "xxxxx-xxxxxxxxxxxx_xxxxxxxxxxxxxxxxxxx";
```

Logical flow here has been improved. At first stage we test if the "*email*" field is not empty then we test it to follow the correct format of email and we check that it originates from a really existing domain. In future, we test if the email account bounces or is notorious for spamming, then IP of the sender becomes blacklisted for our website. For now, those available tests have been moved to lines (**139**) to (**148**).

```
      if( !empty($Email) ){
139                                 list($emailPart, $domainPart) = explode('@',
140 $Email);
141                                 $one=filter_var($Email, FILTER_VALIDATE_EMAIL);
142                                 $two=MailIPAddress($domainPart);
143                                 if (!(($one) && ($two))) {
144                                     $error .= '<li>Email is invalid .</li>';
145                                     echo '<div id="recaptcha_fail_div"> <ul>'.
146 $error .'</ul></div>';
147                                     exit();
148                                 }
                                }
```

For the first time, in line (**144**) our variable $error becomes filled, should the visitor's email not complying our criteria. Pay attention to ***PHP*** operator **.=** which adds the new error to previous value of $error (it was already **null**). This error will be reflected in a formatted box. I have itemised it to be unified with the format of other occurring errors. Here, contact form dispatches the error and exits since it cannot accept the other fields without a valid email. Error box becomes activated by the **echo** command of ***PHP***. That happens in the line (**145**). Please have a look at this image.

Flow of the "*if*" test terminates the job by **exit**() command in the line (**146**); therefore, it does not need any *"else"* phrase.

If the users email looks all right, then flow of the program continues to check other errors ($error at this point still is **null**) for the required fields, through lines (**149**) to (**151**). Each line tests for an empty required field (remember that email already had ***not*** been checked for being empty) and in case will add one item to error box in an itemised format.

```
      $error .= ( empty($Name) )  ? '<li>Please enter your name.</li>' : null;
149   $error .= ( empty($Email) ) ? '<li>Blank Email is not accepted.</li>' :
150 null;
151   $error .= ( empty($Message) ) ? '<li>Please enter some comments or
    questions.</li>' : null;
```

Lines (**153**) to (**157**) get error of reCaptcha field. (It is believed reCaptcha solutions help improve artificial intelligence software that digitise scanned books and documents.) Should reCaptcha field not entered correctly then an error would be added to potential errors.

```
153  $resp = recaptcha_check_answer ($privatekey,
154                                  $_SERVER["REMOTE_ADDR"],
```

```
155                                     $_POST["recaptcha_challenge_field"],
156                                     $_POST["recaptcha_response_field"]);
157  $error .= ( !($resp->is_valid) ) ? '<li>The captcha code entered is
incorrect .</li>' : null;
```

Then we do check whether $error is **null** (is empty) or not. Look at lines (**158**) to (**162**), please,

```
    if (!empty($error)) {
158                      // What happens when the CAPTCHA was entered
159 incorrectly
160                      echo '<div id="recaptcha_fail_div"> <ul>'. $error
161 .'</ul></div>';
162                      exit();
                        }
```

At this stage again we do not need "*else*" for "*if*" since any error terminates flow of the job. Remaining lines are already explained. Result of errors are shown in the following snapshot.

What happens if the form and reCaptcha all filled correctly then you receive your thank you message in the same page. Line (**188**) to line (**193**) guarantee the print of "thanks" message in a

desired format.

```
188 echo '<div style=\'margin-top: 20px; margin-left:112px; border: 2px groove
    blue; border-radius:5px; padding-left: 10px; width:400px;\'>
189             <p>Thanks By <a
    href="http://messiahpsychoanalyst.org">Dysprosium</a></p>
190             <hr style=\'margin: 1px auto 1px auto; height: 1px; color:
    #fefefe; width: 82%;\'/>
191             <h1>Your message has been sent! Thanks for Your Message!</h1>
192             <p><a href="Default.html">Click to go to Home page!</a></p>
193         </div>';
```

Please look at the result.

Please watch this contact form by clicking this link. In the next post I add code to get IP of the visitors and display it on the contact form.

Download ***this tutorial as PDF*** format here