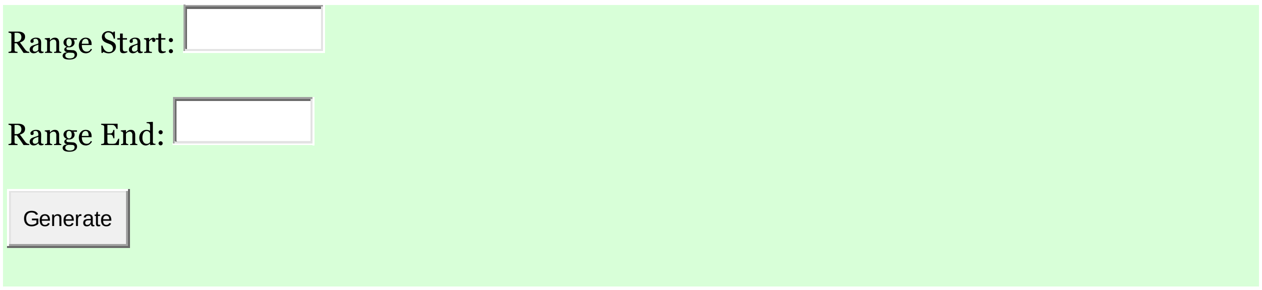
Programming via PHP Accessing user information

4.1. Creating forms

Most interesting server-side Web programs involve reacting to information received from the browser (and ultimately the person using the browser). Before we see how to write PHP to accomplish this, we first examine how to compose HTML to request information from the user using a **form**.

Suppose, for example, that we want to prompt the user for the range of numbers from which the server should select a random number. We decide we want to show the user two different text fields, and a button the user can click to submit the request.



The HTML to generate the above is the following.

```
<form method="post" action="random.php">
<p>Range Start: <input type="text" name="begin" /></p>
<p>Range End: <input type="text" name="end" /></p>
<p><input type="submit" value="Generate" /></p>
</form>
```

This uses two element types specific to forms: `form` and `input`. The first, the `form` element, should encompass the entire form. We'll use two attributes for the `form` tag,`method` and `action`. The `method` attribute configures how the browser should send information back to server; as far as we're concerned, the `method` attribute should always be present and always have the value `post`. The `action` attribute configures what PHP script should receive the information when the user submits the form; this should be the URL of the page you are accessing.

An `input` element places an element as part of the form; normally, this is something with which the user can interact, such as a text field, radio button, checkbox, or button. It has three attributes that are important to use.

- The `type` attribute configures what sort of input element is drawn in the window. Some possible values are `text` for a text field, `radio` for a radio button,`checkbox` for a checkbox, and `submit` for a button.
- The `name` attribute has no effect on how the element is displayed, but when the browser sends information to the server (and ultimately your PHP script), it uses the `name` attribute to tell it what the user has entered; in our above example, if the user enters 10 and 20 into the text fields and then clicks the button, the browser will send a message to the server saying that the user wants to execute the `random.php` script where `begin` has been given the value 10 and `end` has been given the value 20. Without the `name` attribute being specified, we would not be able to access the user's values.
- The `value` attribute configures the initial appearance of the element. For a text field, it says what should originally be in that text field: We omitted it, so the text field initially appears empty; but it might have been nicer to add a `value` attribute for the `begin` field saying that it should be 1 when the user first loads the page, since this is probably the most typical value the user wants to see. For a button, the `value` attribute configures what letters appear in the button.

As a person browsing the Web, you've seen that HTML has ways of incorporating other user interaction elements, such as radio buttons, checkboxes, drop-down choice boxes, file selection areas, lists, and larger text areas. For the moment, though, we'll use just text fields and buttons, which is plenty to be able to compose interesting PHP scripts.

4.2. Reading forms

Suppose the user enters 10 and 20 into the text fields and then clicks the *Generate* button. The browser will send a message to the server saying that the user wants to execute the `random.php` script where `begin` has been given the value 10 and `end` has been given the value 20. We now want to compose `random.php` so that it actually does what the user has requested. Here is an example of how we might write this.

```
<?php import_request_variables("pg", "form_"); ?>
<html>
<head>
<title>Generate Random Number</title>
</head>
```

```
<body>
<p>From the range <?php
    echo $form_begin;
?> to <?php
    echo $form_end;
?> I have selected the random number <?php
    echo rand($form_begin, $form_end);
?>.</p>
</body>
</html>
```

When loaded in response to the above user request, the page might be displayed as:

```
From the range 10 to 20 I have selected the random number 17.
```

You'll notice the first line of the PHP script looks a bit unusual.

```
<?php import_request_variables("pg", "form_"); ?>
```

Don't worry overmuch about exactly what this statement does: You can just type it verbatim at the beginning of each PHP script that is in response to data entered in the form. But as you can see, it invokes the `import_request_variables` function. What this function does is to receive the information given in the form and creates a variable for each value submitted by the user. In particular, for each input value it creates a variable named `$form_`*N*, where *N* stands for the *`name`* attribute associated with that input value. The variable's value will be the value received from the browser (which ultimately is what the user typed into the corresponding input element).

In our running example, our HTML form includes two named *`input`* elements, one named *`begin`* and another named *`end`*, so the `import_request_variables` statement creates two variables named `$form_begin` and `$form_end` whose values are those typed into the corresponding text fields (10 and 20).

The subsequent PHP code is fairly straightforward. We echo back to the user what values the PHP script received, and then we echo a randomly selected number from the specified range.

Source: http://www.toves.org/books/php/ch04-forms/index.html