PHP:Form Series, Part 1: Validators & Client-side Validation

By Dennis Pallett

Welcome to the first part of a new two-part series on the PHP:Form web component. In this part, I will give you an introduction to PHP:Form, its features, and why it is so useful. I will also discuss the inbuilt validators that PHP:Form supports. In the second part I will discuss the more advanced features of PHP:Form.

What is PHP:Form?

PHP:Form is a (new) web component, developed by TPG PHP Scripts. It is designed to help you create forms with a lot less effort. When you're creating a new PHP script, you will undoubtedly have to create forms to allow input to be entered. There is (almost) no PHP script without forms, and forms usually require solid validation to make sure there are no security leaks. It would be nice if those forms are accessible as well, but this often gets forgotten.

We've all had to create forms again and again. And every time we've had to write those validation functions. I'm sure you remember... if (empty($blah)) { echo 'Invalid'; }, etc. After a while it becomes a really boring and mundane task, and most of the time we don't do it properly either. One of the hardest things to do is to return error messages, indicating what is wrong, and allowing visitors to fix their mistakes, without having to re-do the complete form. I used to simply tell them to hit the back-button, after returning the errors, but this is obviously a not-so-good way of doing it.

Thankfully, that's where PHP:Form steps in. It handles all the boring parts for you, and does it properly as well.

PHP:Form has support for inbuilt validators, which means you only have to use a simple HTML-like syntax to add new validation logic to a form. With these validators also comes automatic client-side validation. All the necessary JavaScript is created for you, and there's nothing you need to do. Another remarkable feature of PHP:Form are so-called "formtypes", which are basically form templates. These allow you define form templates, which can be re-used over and over again.

Let's have a closer look at the validation part of PHP:Form; the validators.

Before we look at the validators, let's first look at the basic PHP:Form syntax. It has a really simple syntax, because it's basic HTML. To create a new form, use the
 tags, like so:

```
<php:form name="example">
        ... form html goes here ...
 </php:form>
```

That's all that is really necessary. But you must also tell the form to display, using PHP, like so:

```
$_FORMS->display ('example');
```

Those two things are the only things absolutely necessary to create and display a new form. Of course, nothing will be displayed yet because you haven't created any input fields. To see a simple form in action, have a look at demo 1

Let's move on to validators now. Validators are used to validate form fields, and just like the form tags, they are simple html, for example:

```
<validator for="[fieldname]">Your error message here</validator>
```

You can either place validators in between the form tags, or outside the form tags. If you place them outside the form tags you must specify the form name (using the 'form' attribute).

A simple form with a validator would look like this:

```
<?php

// Include PHP:Form
 include ('../phpform.php');

// Begin form:
 ?>

<php:form name="example">
        <validator for="name" required="true"><p style="font-weight:bold;color:red;">Please enter your name</p></validator>

       Name: <input type="text" name="name" />
        <input type="submit" value="Go!" />
 </php:form>

<?php
 if ($_FORMS->validate ('example') == true) {
        // Show POST'ed values
        echo '<pre>';
        print_r ($_POST);
        echo '</pre>';
 } else {
        // Display form
        $_FORMS->display ("example");
 }
 ?>
```

View live demo

As you can see in the code we created a validator that has the required attribute set to "true". That means that this validator just checks whether the value of the input field isn't empty.

There are 5 different kinds of validators:
 - **Required**: they are used to make sure an input field isn't empty, like I just demonstrated.

```
<validator for="field" required="true">Please fill in something</validator>
```

- **Numeric**: they are used to make sure an input field only contains numbers, and nothing else.

<validator for="field" numeric="true">Please fill in something</validator>

- **Regex**: they can be used to specifiy a regular expression that an input field must match.

<validator for="field" regex="/test/i" >Please enter 'test' only.</validator>

- **Callback**: callback validators can take a callback function that is run on the server-side. That callback function is passed the value of the input field, and the function must return true or false. This is used for really advanced validation (and it's likely you will hardly ever use the callback validator)

<validator for="field" callback="is_email">Not a valid e-mail address.validator>

- **Name**: name validator can be used to display a message or error only when you want to. They can only be shown when you manually show them using the trigger_error ('form', 'errorname') method.

<validator name="mymsg">This is my custom error!</validator>

Then in PHP:

```
<?php
	$_FORMS->trigger_error ('example', 'mymsg');
 ?>
```

When using validators, you will probably want to check if a form validates or not. To do this, use the validate() method, as seen in demo 2:

```
If ($_FORMS->validate('example') == true) {
	echo 'It validates!';
 } else {
	echo It 'doesn\'t validate!';
 }
```

PHP:Form also automatically generates client-side validation (JavaScript) when using validators. It natively supports the required, numeric and regex validators, but it doesn't (fully) support the callback validator. This isn't really possible either, because the callback validator points to a function on the server-side. But if you create a JavaScript function with the same name as the callback function, it will work, and it will run the JavaScript function you created. This gives you great power, and means you can even using advanced JavaScript functions and Ajax to validate data.

If you would like to see the client-side validation in action, have a look at demo 2 again, and make sure you have JavaScript enabled. You will probably notice how fast the errors are returned, and that no refresh happens at all. That's the client-side validation.

Conclusion

In this first part of the PHP:Form series I have shown you what PHP:Form is: an extremely neat PHP form component, that is really useful for building web forms. I have been using it myself now for a few months, and I still can't get over how great it is. It has really simplified things, and I can focus on the important stuff. If you're still in doubt, have a look at the PHP:Form product page for more information and demo's.

I have also shown you exactly what validators are, and the different types. Validators are the most important part of PHP:Form, and you will probably use them in every form. You can some really interesting things with them, and when you combine a few validators it's possible to create a extremely secure form.

In the next part I will have a look at "form types", the form templates of PHP:Form. I will also have a look at setting default values, using the set_value() method of PHP:Form.

If you're interested in purchasing PHP:Form, don't forget to use the special PHPit coupon code: **phpit**
Visit the PHP:Form product page

About this PDF