Validating Form Data

A critical concept related to handling HTML forms is that of validating form data. In terms of both error management and security, you should absolutely never trust the data being entered in an HTML form. Whether erroneous data is purposefully malicious or just unintentionally inappropriate, it's up to you—the Web architect—to test it against expectations.

Validating form data requires the use of conditionals and any number of functions, operators, and expressions. One standard function to be used is `isset()`, which tests if a variable has a value (including *0*, `FALSE`, or an empty string, but not `NULL`). You saw an example of this in the preceding script.

One issue with the `isset()` function is that an empty string tests as `TRUE`, meaning that `isset()` is not an effective way to validate text inputs and text boxes from an HTML form. To check that a user typed something into textual elements, you can use the `empty()` function. It checks if a variable has an *empty* value: an empty string, *0*, `NULL`, or `FALSE`.

The first aim of form validation is seeing if *something* was entered or selected in form elements. The second goal is to ensure that submitted data is of the right type (numeric, string, etc.), of the right format (like an email address), or a specific acceptable value (like `$gender` being equal to either *M* or *F*). As handling forms is a main use of PHP, validating form data is a point that will be re-emphasized time and again in subsequent chapters. But first, let's create a new `handle_form.php` to make sure variables have values before they're referenced (there will be enough changes in this version that simply updating Script 2.3 doesn't make sense).

To validate your forms:

1. Begin a new PHP script in your text editor or IDE (**Script 2.4**).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN" "http:
→ //www.w3.org/TR/xhtml1/DTD/
→ xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/
→ xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
  → content="text/html; charset=
  → iso-8859-1" />
  <title>Form Feedback</title>
</head>
<body>
<?php # Script 2.4 - handle_
→ form.php #3
```

2. Within the HTML `head`, add some CSS (Cascading Style Sheets) code.

```
<style type="text/css" title="text/
→ css" media="all">
.error {
  font-weight: bold;
  color: #C00
}
</style>
```

CSS is the preferred way to handle many formatting and layout issues in an HTML page. You'll see a little bit of CSS here and there in this book; if you're not familiar with the subject, check out a dedicated CSS reference.

continues on page 49

Script 2.4 Validating HTML form data before you use it is critical to Web security and achieving professional results. Here, conditionals check that every referenced form element has a value.

Script

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
  <title>Form Feedback</title>
  <style type="text/css" title="text/css" media="all">
  .error {
     font-weight: bold;
     color: #C00
  }
  </style>
</head>
<body>
<?php # Script 2.4 - handle_form.php #3

// Validate the name:
if (!empty($_REQUEST['name'])) {
  $name = $_REQUEST['name'];
} else {
  $name = NULL;
  echo '<p class="error">You forgot to enter your name!</p>';
}

// Validate the email:
if (!empty($_REQUEST['email'])) {
  $email = $_REQUEST['email'];
} else {
  $email = NULL;
  echo '<p class="error">You forgot to enter your email address!</p>';
}

// Validate the comments:
if (!empty($_REQUEST['comments'])) {
  $comments = $_REQUEST['comments'];
} else {
  $comments = NULL;
```

(script continues on next page)

Script 2.4 *continued*

```
    echo '<p class="error">You forgot to enter your comments!</p>';
}

// Validate the gender:
if (isset($_REQUEST['gender'])) {

    $gender = $_REQUEST['gender'];

    if ($gender == 'M') {
        echo '<p><b>Good day, Sir!</b></p>';
    } elseif ($gender == 'F') {
        echo '<p><b>Good day, Madam!</b></p>';
    } else { // Unacceptable value.
        $gender = NULL;
        echo '<p class="error">Gender should be either "M" or "F"!</p>';
    }

} else { // $_REQUEST['gender'] is not set.
    $gender = NULL;
    echo '<p class="error">You forgot to select your gender!</p>';
}

// If everything is OK, print the message:
if ($name && $email && $gender && $comments) {

    echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
    <tt>$comments</tt></p>
    <p>We will reply to you at <i>$email</i>.</p>\n";

} else { // Missing form value.
    echo '<p class="error">Please go back and fill out the form again.</p>';
}

?>
</body>
</html>
```

In this script I'm defining one CSS class, called *error*. Any HTML element that has this class name will be formatted in a bold, red color (which will be more apparent in your Web browser than in this black-and-white book).

3. Check if the name was entered.

```
if (!empty($_REQUEST['name'])) {
  $name = $_REQUEST['name'];
} else {
  $name = NULL;
  echo '<p class="error">You forgot
  → to enter your name!</p>';
}
```

A simple way to check that a form text input was filled out is to use the `empty()` function. If `$_REQUEST['name']` has a value other than an empty string, *0*, `NULL`, or `FALSE`, assume that their name was entered and a shorthand variable is assigned that value. If `$_REQUEST['name']` is empty, the `$name` variable is set to `NULL` and an error message printed. This error message uses the CSS class.

4. Repeat the same process for the email address and comments.

```
if (!empty($_REQUEST['email'])) {
  $email = $_REQUEST['email'];
} else {
  $email = NULL;
  echo '<p class="error">You forgot
to enter your email address!</p>';
}
if (!empty($_REQUEST['comments'])) {
  $comments = $_REQUEST['comments'];
} else {
  $comments = NULL;
  echo '<p class="error">You forgot
  → to enter your comments!</p>';
}
```

Both variables receive the same treatment as `$_REQUEST['name']` in Step 3.

5. Begin validating the gender variable.

```
if (isset($_REQUEST['gender'])) {
  $gender = $_REQUEST['gender'];
```

The validation of the gender is a two-step process. First, check if it has a value or not, using `isset()`. This starts the main `if-else` conditional, which otherwise behaves like those for the name, email address, and comments.

6. Check `$gender` against specific values.

```
if ($gender == 'M') {
  echo '<p><b>Good day, Sir!</b>
  → </p>';
} elseif ($gender == 'F') {
  echo '<p><b>Good day, Madam!</b>
  → </p>';
} else {
  $gender = NULL;
  echo '<p class="error">Gender
  → should be either "M" or "F"!
  → </p>';
}
```

Within the gender `if` clause is a nested `if-elseif-else` conditional that tests the variable's value against what's acceptable. This is the second part of the two-step gender validation.

continues on next page

The conditions themselves are the same as those in the last script. If gender does not end up being equal to either *M* or *F*, a problem occurred and an error message is printed. The `$gender` variable is also set to `NULL` in such cases, because it has an unacceptable value.

If `$gender` does have a valid value, a gender-specific message is printed.

7. Complete the main gender `if-else` conditional.

```
} else {
  $gender = NULL;
  echo '<p class="error">You forgot
  → to select your gender!</p>';
}
```

This `else` clause applies if `$_REQUEST ['gender']` is not set. The complete, nested conditionals (see lines 41–57 of Script 2.4) successfully check every possibility:

▲ `$_REQUEST['gender']` is not set

▲ `$_REQUEST['gender']` has a value of *M*

▲ `$_REQUEST['gender']` has a value of *F*

▲ `$_REQUEST['gender']` has some other value

You may wonder how this last case may be possible, considering the values are established in the HTML form. If a malicious user creates their own form that gets submitted to your `handle_form.php` script (which is very easy to do), they could give `$_REQUEST['gender']` any value they want.

8. Print the message if all of the tests have been passed.

```
if ($name && $email && $gender &&
→ $comments) {
  echo "<p>Thank you, <b>$name</b>,
  → for the following comments:
  → <br />
  <tt>$comments</tt></p>
  <p>We will reply to you at <i>$
  → email</i>.</p>\n";
} else { // Missing form value.
  echo '<p class="error">Please go
  → back and fill out the form
  → again.</p>';
}
```

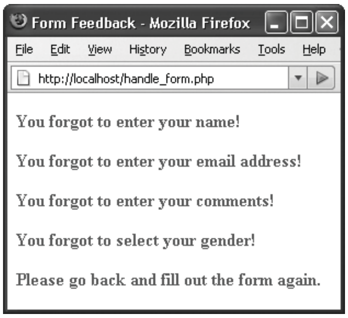
This main condition is true if every listed variable has a true value. Each variable will have a value if it passed its test but have a value of `NULL` if it didn't. If every variable has a value, the form was completed, so the *Thank you* message will be printed. If any of the variables are `NULL`, the second message will be printed (**Figures 2.13** and **2.14**).

9. Close the PHP section and complete the HTML code.

```
?>
</body>
</html>
```

10. Save the file as `handle_form.php`, place it in the same Web directory as `form.html`, and test it in your Web browser (Figures 2.13 and 2.14).

Fill out the form to different levels of completeness to test the new script (**Figure 2.15**).


Form Feedback - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/handle_form.php

You forgot to enter your name!

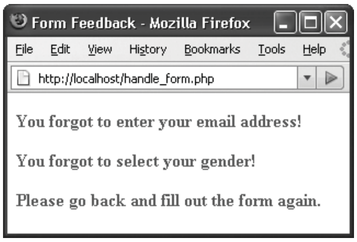
You forgot to enter your email address!

You forgot to enter your comments!

You forgot to select your gender!

Please go back and fill out the form again.

Figure 2.13 The script now checks that every form element was filled out (except the age) and reports on those that weren't.


Form Feedback - Mozilla Firefox

File Edit View History Bookmarks Tools Help

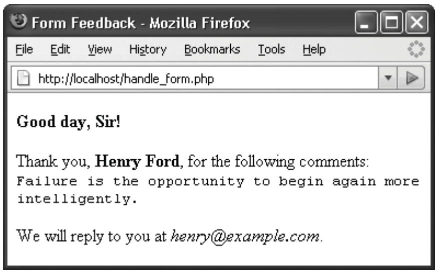
http://localhost/handle_form.php

You forgot to enter your email address!

You forgot to select your gender!

Please go back and fill out the form again.

Figure 2.14 If even one or two fields were skipped, the *Thank you* message is not printed...


Form Feedback - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/handle_form.php

Good day, Sir!

Thank you, **Henry Ford**, for the following comments:
```
Failure is the opportunity to begin again more
intelligently.
```
We will reply to you at *henry@example.com*.

Figure 2.15 ...but if everything was entered properly, the script behaves as it previously had (although the gender-specific message now appears at the top of the results).

✔ Tips

- To test if a submitted value is a number, use the `is_numeric()` function.
- In Chapter 13, "Perl-Compatible Regular Expressions," you'll see how to validate form data using regular expressions.
- The `$age` variable is still not used or validated for the sake of saving book space. To validate it, repeat the `$gender` validation routine, referring to `$_REQUEST['age']` instead. To test `$age`'s specific value, use an `if-elseif-elseif-else`, checking against the corresponding pull-down options (*0-29*, *30-60*, *60+*).
- It's considered good form (pun intended) to let a user know which fields are required when they're filling out the form, and where applicable, the format of that field (like a date or a phone number).