Beginning Ajax with PHP

From Novice to Professional

Lee Babin

Apress®

Beginning Ajax with PHP: From Novice to Professional

ISBN-13 (pbk): 978-1-59059-667-8

ISBN-10 (pbk): 1-59059-667-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jason Gilmore
Technical Reviewer: Quentin Zervaas
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Richard Dal Porto
Copy Edit Manager: Nicole Flores
Copy Editors: Damon Larson, Jennifer Whipple
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Dina Quan
Proofreader: Lori Bring
Indexer: John Collin
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

CHAPTER 3

PHP and Ajax

While the concept of Ajax contains a handy set of functionality for creating actions on the fly, if you are not making use of its ability to connect to the server, you are really just using basic JavaScript. Not that there is anything truly wrong with that, but the real power lies in joining the client-side functionality of JavaScript with the server-side processing of the PHP language using the concept of Ajax.

Throughout this chapter, I will run through some examples of how PHP and Ajax can be used together to design some basic tools that are quite new to Internet applications but have been accessible to desktop applications for ages. The ability to make a call to the server without a page refresh is one that is quite powerful, if harnessed correctly. With the help of the powerful PHP server-side language, you can create some handy little applications that can be easily integrated into any web project.

Why PHP and Ajax?

So, out of all of the available server-side processing languages (ASP, ASP.NET, ColdFusion, etc.), why have I chosen to devote this book to the PHP language, as any of them can function decently with Ajax technologies? Well, the truth is that while any of the aforementioned languages will perform admirably with Ajax, PHP has similarities with the JavaScript language used to control Ajax—in functionality, code layout, and ideology.

PHP has been and will likely continue to be a very open form of technology. While code written in PHP is always hidden from the web user, there is a massive community of developers who prefer to share and share alike when it comes to their code. You need only scour the web to find an abundance of examples, ranging from the most basic to the most in-depth. When comparing PHP's online community against a coding language such as ASP.NET, it is not difficult to see the differences.

JavaScript has always been an open sort of technology, largely due to the fact that it does not remain hidden. Because it is a client-side technology, it is always possible to view the code that has been written in JavaScript. Perhaps due to the way JavaScript is handled in this manner, JavaScript has always had a very open community as well. By combining the communities of JavaScript and PHP, you can likely always find the examples you want simply by querying the community.

To summarize why PHP and Ajax work so well together, it comes down to mere functionality. PHP is a very robust, object-oriented language. JavaScript is a rather robust language in itself; it is sculptured after the object-oriented model as well. Therefore, when you combine two languages, aged to maturity, you come away with the best of both worlds, and you are truly ready to begin to merge them for fantastic results.

Client-Driven Communication, Server-Side Processing

As I have explained in previous chapters, there are two sides to a web page's proverbial coin. There is the client-side communication aspect—that is, the functionality happening right then and there on the client's browser; and the server-side processing—the more intricate levels of scripting, which include database interaction, complex formulas, conditional statements, and much, much more.

For the entirety of this book, you will be making use of the JavaScript language to handle the client-side interaction and merging it seamlessly with the PHP processing language for all your server-side manipulation. By combining the two, the sky is truly the limit. Anything that can be imagined can come to fruition if enough creativity and hard work is put into it.

Basic Examples

In order to get geared up for some of the more intricate and involved examples, I will begin by showing some basic examples of common web mini-applications that work well with the Ajax ideology. These are examples you are likely to see already in place in a variety of web applications, and they are a very good basis for showing what can be accomplished using the Ajax functionality.

Beyond the fact that these applications have become exceedingly popular, this chapter will attempt to guide you as to what makes these pieces of functionality so well-suited to the Ajax concept. Not every application of Ajax is necessarily a good idea, so it is important to note why these examples work well with the Ajax concept, and how they make the user's web-browsing experience better. What would the same application look like if the page had to refresh? Would the same functionality have even been possible without Ajax, and how much work does it save us (if any)?

Expanding and Contracting Content

One spectacular use for Ajax-type functionality is in hiding content away and exposing it based on link clicks (or hovers, or button presses). This sort of functionality allows you to

create access to a large amount of content without cluttering the screen. By hiding content within expandable and retractable menu links, you can add a lot of information in a small amount of space.

Consider the following example, which uses Ajax to expand and contract a calendar based upon link clicks. By using Ajax to hide and show information, and PHP to dynamically generate a calendar based upon the current month, you create a well-hidden calendar that can be added to any application with relative ease and very little web site real estate.

In order to start things off, you need a valid web page in which to embed your calendar. The following code will create your very basic web page:

```
<!-- sample3_1.html -->
<!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">
<head>
<title>Sample 3_1</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<script type="text/javascript" src="functions.js"></script>
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
  <div id="createtask" class="formclass"></div>
  <div id="autocompletediv" class="autocomp"></div>
  <div id="taskbox" class="taskboxclass"></div>
  <p><a href="javascript://" onclick="showHideCalendar()">➥
<img id="opencloseimg" src="images/plus.gif" alt="" title="" ➥
style="border: none; width: 9px; height: 9px;" /></a>➥
 <a href="javascript://" onclick="showHideCalendar()">My Calendar</a></p>
  <div id="calendar" style="width: 105px; text-align: left;"></div>
</body>
</html>


  //functions.js

  //Create a boolean variable to check for a valid IE instance.
  var xmlhttp = false;
```

```
//Check if we are using IE.
try {
  //If the javascript version is greater than 5.
  xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  //If not, then use the older active x object.
  try {
    //If we are using IE.
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (E) {
    //Else we must be using a non-IE browser.
    xmlhttp = false;
  }
}

//If we are using a non-IE browser, create a JavaScript instance of the object.
if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
  xmlhttp = new XMLHttpRequest();
}

//A variable used to distinguish whether to open or close the calendar.
var showCalendar = true;

function showHideCalendar() {

  //The location we are loading the page into.
  var objID = "calendar";

  //Change the current image of the minus or plus.
  if (showCalendar == true){
    //Show the calendar.
    document.getElementById("opencloseimg").src = "images/mins.gif";
    //The page we are loading.
    var serverPage = "calendar.php";
    //Set the open close tracker variable.
    showCalendar = false;

    var obj = document.getElementById(objID);
    xmlhttp.open("GET", serverPage);
    xmlhttp.onreadystatechange = function() {
```

```
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        obj.innerHTML = xmlhttp.responseText;
      }
    }
    xmlhttp.send(null);
  } else {
    //Hide the calendar.
    document.getElementById("opencloseimg").src = "images/plus.gif";
    showCalendar = true;

    document.getElementById(objID).innerHTML = "";
  }


}
```

This looks fairly basic, right? What you need to take into account is the JavaScript contained within the `functions.js` file. A function called `showHideCalendar` is created, which will either show or hide the calendar module based upon the condition of the `showCalendar` variable. If the `showCalendar` variable is set to `true`, an Ajax call to the server is made to fetch the `calendar.php` script. The results from said script are then displayed within the `calendar` page element. You could obviously modify this to load into whatever element you prefer. The script also changes the state of your plus-and-minus image to show true open-and-close functionality.

Once the script has made a call to the server, the PHP script will use its server-side functionality to create a calendar of the current month. Consider the following code:

```
<?php

  //calendar.php

  //Check if the month and year values exist
  if ((!$_GET['month']) && (!$_GET['year'])) {
    $month = date ("n");
    $year = date ("Y");
  } else {
    $month = $_GET['month'];
    $year = $_GET['year'];
  }
```

```
  //Calculate the viewed month
  $timestamp = mktime (0, 0, 0, $month, 1, $year);
  $monthname = date("F", $timestamp);

  //Now let's create the table with the proper month
  ?>
  <table style="width: 105px; border-collapse: collapse;" border="1"➥
 cellpadding="3" cellspacing="0" bordercolor="#000000">
    <tr style="background: #FFBC37;">
    <td colspan="7" style="text-align: center;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;"><?php echo $monthname➥
 . " " .  $year; ?></span>
    </td>
    </tr>
    <tr style="background: #FFBC37;">
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">Su</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">M</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">Tu</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">W</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">Th</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
      <span style="font-weight: bold;">F</span>
    </td>
    <td style="text-align: center; width: 15px;" onmouseover=➥
"this.style.background='#FECE6E'" onmouseout="this.style.background='#FFBC37'">
```

```
      <span style="font-weight: bold;">Sa</span>
    </td>
    </tr>
    <?php
      $monthstart = date("w", $timestamp);
      $lastday = date("d", mktime (0, 0, 0, $month + 1, 0, $year));
      $startdate = -$monthstart;

      //Figure out how many rows we need.
      $numrows = ceil (((date("t",mktime (0, 0, 0, $month + 1, 0, $year))➥
 + $monthstart) / 7));

      //Let's make an appropriate number of rows...
      for ($k = 1; $k <= $numrows; $k++){
        ?><tr><?php
        //Use 7 columns (for 7 days)...
        for ($i = 0; $i < 7; $i++){
          $startdate++;
          if (($startdate <= 0) || ($startdate > $lastday)){
            //If we have a blank day in the calendar.
            ?><td style="background: #FFFFFF;"> </td><?php
          } else {
            if ($startdate == date("j") && $month == date("n") &&➥
 $year == date("Y")){
              ?><td style="text-align: center; background: #FFBC37;" ➥
 onmouseover="this.style.background='#FECE6E'"➥
 onmouseout="this.style.background='#FFBC37'">➥
<?php echo date ("j"); ?></td><?php
            } else {
              ?><td style="text-align: center; background: #A2BAFA;" ➥
 onmouseover="this.style.background='#CAD7F9'"➥
 onmouseout="this.style.background='#A2BAFA'">➥
<?php echo $startdate; ?></td><?php
            }
          }
        }
        ?></tr><?php
      }
      ?>
    </table>
```

This is simply code to show a calendar of the current month. The code is set up to allow for alternative years and months, which can be passed in with the `$_GET` super-global; but for now, you are going to concentrate only on the current month. As you progress with the examples in this chapter, you will see how you can use Ajax to really improve the functionality of this module and create some very cool applications.

The code itself is fairly simple to decipher. It simply uses the `date` function in PHP to determine the beginning and end dates, and then build the calendar accordingly. This is a prime example of using PHP's server-side scripting in conjunction with Ajax to create a nice little application (as shown in Figure 3-1). Next, you'll work on progressing your application.

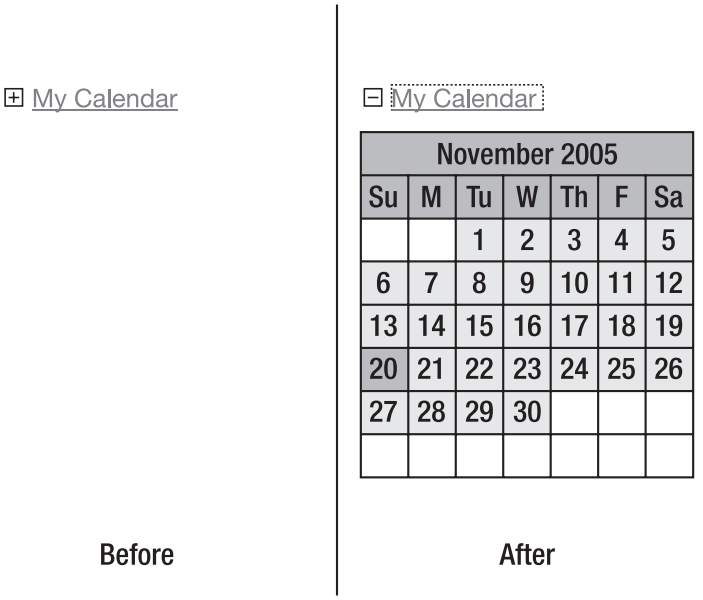

Figure 3-1. *The calendar application pulls an appearing/disappearing act.*

Auto-Complete

A nice feature that I first noticed as being received positively by the Internet community is the auto-complete feature in Gmail. Basically, when you're entering the e-mail address of the person you're sending a message to, Gmail searches your list of contacts (using Ajax) and automatically drops down a listing of all matches. You are then free to click one of the dropped-down objects to fill it into the requested field. The whole code integration is seamless and makes for a handy feature.

The next example will show you how to do the same thing—although it's not quite as in-depth as the Gmail solution. Basically, I have built a way to feed a list of objects

through an array (a database solution would be more effective, but that is outside of the scope of this example and will be shown later in the book), and then display them based on what the user has entered. The user can then click the name to fill out the field (hence the auto-completion).

I have expanded on the previous example by assuming that a user may want to enter a reminder for the particular day in question on the calendar. The system allows the user to enter their name and their task by clicking on an individual day. Ideally, once the task is entered, the system will then save the task to the database. For now, though, you are merely concentrating on the auto-complete feature; saving the actual information will be handled in a later chapter.

Have a look at the following example, which integrates an auto-complete feature and a pop-up form using Ajax. Pay attention to the `style.css` and `functions.js` files, which have changed.

```
/* style.css */

body {
  font-family: verdana, arial, helvetica;
  font-size: 11px;
  color: #000000;
}

.formclass {
  position: absolute;
  left: 0px;
  top: 0px;
  visibility: hidden;
  height: 0px;
  width: 0px;
  background: #A2BAFA;
  border-style: solid;
  border-width: 1px;
  border-color: #000000;
}

.autocomp {
  position: absolute;
  left: 0px;
  top: 0px;
  visibility: hidden;
  width: 0px;
}
```

```
.taskboxclass {
  position: absolute;
  left: 0px;
  top: 0px;
  visibility: hidden;
  width: 0px;
}


.calendarover {
  text-align: center;
  background: #CAD7F9;
  width: 15px;
}

.calendaroff {
  text-align: center;
  background: #A2BAFA;
  width: 15px;
}

.calendartodayover {
  text-align: center;
  background: #FECE6E;
  width: 15px;
}

.taskchecker {
  width: 150px;
  background-color: #FFBC37;
  border-style: solid;
  border-color: #000000;
  border-width: 1px;
}
```

```
.tcpadding {
  padding: 10px;
}

.calendartodayoff {
  text-align: center;
  background: #FFBC37;
  width: 15px;
}


  //functions.js

  function createform (e){

    theObject = document.getElementById("createtask");

    theObject.style.visibility = "visible";
    theObject.style.height = "200px";
    theObject.style.width = "200px";

    var posx = 0;
    var posy = 0;

    posx = e.clientX + document.body.scrollLeft;
    posy = e.clientY + document.body.scrollTop;

    theObject.style.left = posx + "px";
    theObject.style.top = posy + "px";

    //The location we are loading the page into.
    var objID = "createtask";
    var serverPage = "theform.php";
```

```
  var obj = document.getElementById(objID);
  xmlhttp.open("GET", serverPage);
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      obj.innerHTML = xmlhttp.responseText;
    }
  }
  xmlhttp.send(null);

}

function closetask (){

  theObject = document.getElementById("createtask");

  theObject.style.visibility = "hidden";
  theObject.style.height = "0px";
  theObject.style.width = "0px";

  acObject = document.getElementById("autocompletediv");

  acObject.style.visibility = "hidden";
  acObject.style.height = "0px";
  acObject.style.width = "0px";
}

function findPosX(obj){
  var curleft = 0;
  if (obj.offsetParent){
    while (obj.offsetParent){
      curleft += obj.offsetLeft
      obj = obj.offsetParent;
    }
  } else if (obj.x){
    curleft += obj.x;
  }
  return curleft;
}
```

```
function findPosY(obj){
  var curtop = 0;
  if (obj.offsetParent){
    while (obj.offsetParent){
      curtop += obj.offsetTop
      obj = obj.offsetParent;
    }
  } else if (obj.y){
    curtop += obj.y;
  }
  return curtop;
}

function autocomplete (thevalue, e){

  theObject = document.getElementById("autocompletediv");

  theObject.style.visibility = "visible";
  theObject.style.width = "152px";

  var posx = 0;
  var posy = 0;

  posx = (findPosX (document.getElementById("yourname")) + 1);
  posy = (findPosY (document.getElementById("yourname")) + 23);

  theObject.style.left = posx + "px";
  theObject.style.top = posy + "px";

  var theextrachar = e.which;

  if (theextrachar == undefined){
    theextrachar = e.keyCode;
  }

  //The location we are loading the page into.
  var objID = "autocompletediv";
```

```
    //Take into account the backspace.
    if (theextrachar == 8){
      if (thevalue.length == 1){
        var serverPage = "autocomp.php";
      } else {
        var serverPage = "autocomp.php" + "?sstring=" + ➥
thevalue.substr (0, (thevalue.length -1));
      }
    } else {
      var serverPage = "autocomp.php" + "?sstring=" + ➥
thevalue + String.fromCharCode (theextrachar);
    }

    var obj = document.getElementById(objID);
    xmlhttp.open("GET", serverPage);
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        obj.innerHTML = xmlhttp.responseText;
      }
    }
    xmlhttp.send(null);
  }

  function setvalue (thevalue){
    acObject = document.getElementById("autocompletediv");

    acObject.style.visibility = "hidden";
    acObject.style.height = "0px";
    acObject.style.width = "0px";

    document.getElementById("yourname").value = thevalue;
  }
```

Now, let's have a look at what has changed since the last example. Quite a number of functions have been added. The first is called `createform`. The `createform` function displays a hidden `div` beside where the cursor is currently located, and then loads in a file called `theform.php` through Ajax. This function uses mostly JavaScript to get the job done (through `hidden` and `visible` style aspects), but Ajax comes into play to load the file. The code for the `theform.php` file (basically a simple entry form) is shown in the following snippet:

```
<!-- theform.php -->
<div style="padding: 10px;">
  <div id="messagebox"></div>
  <form action="" method="post">
    Your Name<br />
    <input id="yourname" style="width: 150px; height: 16px;"➥
 type="text" value="" onkeypress="autocomplete(this.value, event)" /><br />
    Your Task<br />
    <textarea style="height: 80px;"></textarea><br />
    <div align="right"><a href="javascript:closetask()">close</a></div>
  </form>
</div>
```

The next set of functions mostly do cleanup work and fetch requests. The `closetask` function "closes," or effectively hides the task window should the user decide they no longer wish to enter a task. The `findPosX` and `findPosY` functions return the current x and y positions of the field you want to assign the auto-complete functionality to.

The next two functions, `autocomplete` and `setvalue`, are the two that do the actual auto-complete. Basically, the function `autocomplete` checks for the currently inputted string (using the `onkeypress` event) and passes said string to a file called `autocomp.php` via Ajax. There is quite a bit of code in place to make this function as browser-compliant as possible—dealing with key presses and events from browser to browser can be tricky.

The important matter is that a string representing the current value of the text box (the Your Name field) is passed to a PHP file on the fly. The next block of code displays what the PHP script does with the passed-in information.

```
<?php

  //A list of all names.
  //Generally this would be in a database of some sort.
  $names = array ("Lee Babin","Joe Smith","John Doe");
  $foundarr = array ();

  //Go through the names array and load any matches into the foundarr array.
  if ($_GET['sstring'] != ""){
    for ($i = 0; $i < count ($names); $i++){
      if (substr_count (strtolower ($names[$i]), ➥
strtolower ($_GET['sstring'])) > 0){
        $foundarr[] = $names[$i];
      }
    }
  }
```

```
  //If we have any matches.
  if (count ($foundarr) > 0){
    //Then display them.
    ?>
    <div style="background: #CCCCCC; border-style: solid; ➥
border-width: 1px; border-color: #000000;">
      <?php
        for ($i = 0; $i < count ($foundarr); $i++){
          ?><div style="padding: 4px; height: 14px;" onmouseover=➥
"this.style.background = '#EEEEEE'" onmouseout=➥
"this.style.background = '#CCCCCC'" onclick=➥
"setvalue ('<?php echo $foundarr[$i]; ?>')"><?php echo $foundarr[$i]; ?> ➥
</div><?php
        }
      ?>
    </div>
    <?php
  }

?>
```

The `autocomp.php` file takes the passed-in string and attempts to find any matches. As it finds valid matches to the query string, it loads them into another array. The reason for loading into an alternate array is to keep the height of the `div` at nothing unless a valid match has been found. If a valid match (or set of matches) is acquired, the auto-complete shows the correct matches. If you are to click a valid match, it will load the name into the appropriate form field (using the `setvalue` function) and close the auto-complete pop-up form. Voilà, you now have a fully functioning auto-complete feature using Ajax technology (as shown in Figure 3-2).

Not only does this feature save the user a large amount of time, it just feels very intuitive. It is important to make applications as simple as possible so that newly initiated Internet users find it easy to get along with your applications.

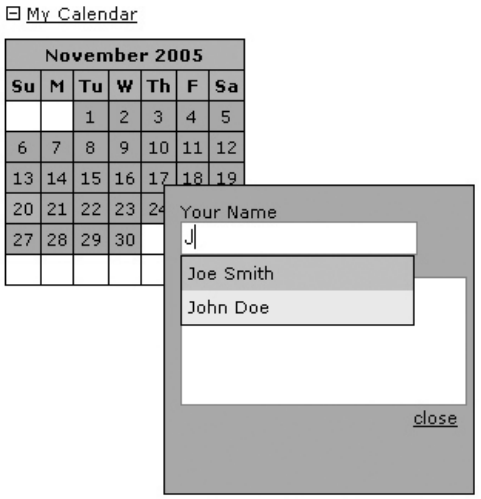

Figure 3-2. *Auto-complete makes data entry seamless and effective.*

Form Validation

I won't get too far into form validation until Chapter 5, when I discuss forms in their entirety. However, it would be prudent to show a rather nice trick that can be done using Ajax to validate what used to be a difficult set of information to error check. Most fields could be validated on the client side by using JavaScript to determine empty fields, bad information, and so on. There was, however, always a problem with checking for valid information that might be contained within a database that only your scripting language could identify.

Now that you have Ajax as a tool, however, you can get the best of both worlds. The workaround in the past was to submit the form, check the server, send back all values that were currently entered, and prepopulate the form when the screen returned. While this worked fairly well, it was a rather large task to code all of it, and it did not work with such fields as file uploads (which cannot be prepopulated). In the next example, you will use the same task-entry code as you used earlier, but now when you submit the form, you will first check whether the Your Name field exists within your script before allowing submittal. This simulates something like a username validator, in which a user is prevented from entering a username that is already taken when signing up at a site.

Rather than show the entire code set over again, let's go over changes that have been made. First off, I have added a new function called `validateform`, as shown in the following code:

```
//functions.js
function validateform (thevalue){

  serverPage = "validator.php?sstring=" + thevalue;
  objID = "messagebox";

  var obj = document.getElementById(objID);
  xmlhttp.open("GET", serverPage);
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      obj.innerHTML = xmlhttp.responseText;
    }
  }
  xmlhttp.send(null);
}
```

This function loads a PHP script into a certain section of your page. The following code contains the changes to the form:

```
<!-- theform.php -->
<div style="padding: 10px;">
  <div id="messagebox"></div>
    <form method="post">
      Your Name<br />
      <input id="yourname" style="width: 150px; height: 16px;"➥
 type="text" value="" onkeypress="autocomplete(this.value, event)" />➥
<br />
      Your Task<br />
      <textarea style="height: 80px;"></textarea><br />
      <input type="button" value="Submit" onclick="validateform➥
 (document.getElementById('yourname').value)" />
  <div align="right"><a href="javascript:closetask()">close</a></div>
  </form>
</div>
```

As you can see, you have added a `div` called `messagebox` (which will show any errors you may come across) and a button that you are using to call the `validateform` function. When that button is clicked, the `validateform` function will fire, accessing a PHP script contained within a file called `validator.php`. The code for this is shown following:

```
<?php
  //validator.php

  //A list of valid names.
  //Again, this would usually come from a database.
  $names = array ("Lee Babin","Joe Smith","John Doe");

  if (!in_array (strtolower ($_GET['sstring']), strtolower ($names))){
    //Then return with an error.
    ?><span style="color: #FF0000;">Name not found...</span><?php
  } else {
    //At this point we would go to the processing script.
    ?><span style="color: #FF0000;">Form would now submit...</span><?php
  }
?>
```

All the PHP script does is check for a valid match from the passed-in Your Name field. If a match is found, the script would merely submit the form using JavaScript (in this case, it merely displays a message—I will discuss more on submitting a form using JavaScript later in this book). If an error is found, you can output the error seamlessly and rather quickly. The nice thing about this little bit of functionality is that your form stays populated (since the form has not been submitted yet). This saves you a lot of time from a coding perspective and makes things seamless and intuitive for the user (see Figure 3-3).

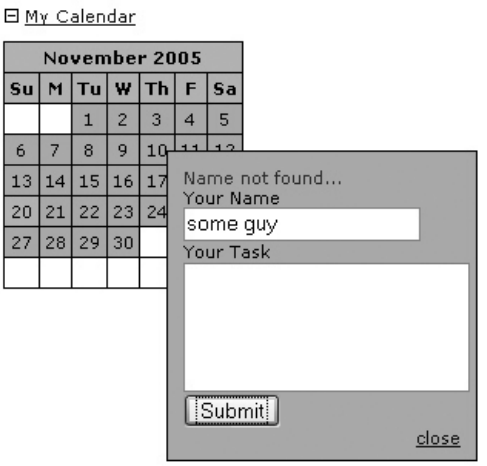

Figure 3-3. *As you can see, names that are not supposed to be entered can be validated against.*

Tool Tips

One of the more common DHTML "tricks" you will see on the Internet is the tool tip. This is basically a little box filled with information that will appear above a properly placed cursor. While this is all fine and dandy, the information contained within said box in non-Ajax enabled web sites is usually either hard-coded in or potentially loaded through some server-side trickery. What you will do in the next example is load the box dynamically using Ajax.

As a useful addition to your calendar application, it would be nice to dynamically display a box with all currently assigned tasks when a user hovers over a certain date. The PHP script would henceforth have to scour the database and return any instances of tasks associated with said day. Since I'm not going to get into databases just yet, I'll have you build the script to accommodate an array of tasks for now, just to showcase the tool tip functionality.

First off, let's have a look at the `calendar.php` file in order to view the changes to the calendar code:

```
  //Let's make an appropriate number of rows...
  for ($k = 1; $k <= $numrows; $k++){
    ?><tr><?php
    //Use 7 columns (for 7 days)...
    for ($i = 0; $i < 7; $i++){
      $startdate++;
      if (($startdate <= 0) || ($startdate > $lastday)){
        //If we have a blank day in the calendar.
        ?><td style="background: #FFFFFF;"> </td><?php
      } else {
        if ($startdate == date("j") && $month == date("n") && $year == date("Y")){
          <td onclick="createform(event)" class="calendartodayoff"➥
 onmouseover="this.className='calendartodayover'; checkfortasks ➥
('<?php echo $year . "-" . $month . "-" . $startdate; ?>',event);"➥
 onmouseout="this.className='calendartodayoff'; hidetask();">➥
<?php echo date ("j"); ?></td><?php
        } else {
          <td onclick="createform(event)" class="calendaroff"➥
 onmouseover="this.className='calendarover'; checkfortasks➥
 ('<?php echo $year . "-" . $month . "-" . $startdate; ?>',event);" ➥
onmouseout="this.className='calendaroff'; hidetask();">➥
<?php echo $startdate; ?></td><?php
        }
      }
    }
    ?></tr><?php
  }
```

The major change made here is calling a `checkfortasks` function that is fired by the `onmouseover` event, and a `hidetask` function that fires on the `onmouseout` event. Basically, the current date that a user is hovering over will be passed to the appropriate functions, which are located within the `functions.js` file (shown following):

```
//functions.js
function checkfortasks (thedate, e){

  theObject = document.getElementById("taskbox");

  theObject.style.visibility = "visible";

  var posx = 0;
  var posy = 0;

  posx = e.clientX + document.body.scrollLeft;
  posy = e.clientY + document.body.scrollTop;

  theObject.style.left = posx + "px";
  theObject.style.top = posy + "px";

  serverPage = "taskchecker.php?thedate=" + thedate;
  objID = "taskbox";

  var obj = document.getElementById(objID);
  xmlhttp.open("GET", serverPage);
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      obj.innerHTML = xmlhttp.responseText;
    }
  }
  xmlhttp.send(null);
}

function hidetask (){
  tObject = document.getElementById("taskbox");

  tObject.style.visibility = "hidden";
  tObject.style.height = "0px";
  tObject.style.width = "0px";
}
```

Again, your tool tip machine uses some DHTML tricks to hide the `div` you want to load your task-checker script within. You will need to create the new `div` as shown in the following code in order for this to work properly.

```
<body>
  <div id="createtask" class="formclass"></div>
  <div id="autocompletediv" class="autocomp"></div>
  <div id="taskbox" class="taskboxclass"></div>
  <p><a href="javascript://" onclick="showHideCalendar()"><img id="opencloseimg"➥
 src="images/plus.gif" alt="" title"" style="border: none;➥
 width: 9px; height: 9px;" /></a> <a href="javascript://" onclick=➥
"showHideCalendar()">My Calendar</a></p>
  <div id="calendar" style="width: 105px; text-align: left;"></div>
</body>
```

The `checkfortasks` function will accept a date and then pass it along (via Ajax) to a new file called `taskchecker.php`. The `taskchecker.php` file would then usually read from a database and show the appropriate tasks for the current date in a dynamically created, hovering `div` (not unlike the task entry form). In this case, because you don't want to get into database integration just yet, you have made use of an associative array. The code for `taskchecker.php` is as follows:

```
<?php
  //taskchecker.php
  //Actual Task dates.
  //This would normally be loaded from a database.
  $tasks = array ("2005-11-10" => 'Check mail.',"2005-11-20" => 'Finish Chapter 3');

  $outputarr = array ();

  //Run through and check for any matches.
  while ($ele = each ($tasks)){
    if ($ele['key'] == $_GET['thedate']){
      $outputarr[] = $ele['value'];
    }
  }
```

```
  //If we have any matches...
  if (count ($outputarr) > 0){
    ?>
    <div class="taskchecker">
      <div class="tcpadding">
      <?php
        for ($i = 0; $i < count ($outputarr); $i++){
          echo $outputarr[$i] . "<br />";
        }
      ?>
      </div>
    </div>
    <?php
  }
?>
```

As you can see, the system runs through the associative array (this would normally be a database query) and then loads any matches into the `outputarr` array variable. Then, if any matches are found, it displays them within a `div` that you create on the fly. The result is a very dynamic task listing, as shown in Figure 3-4.

Figure 3-4. *Displaying tasks through the magic of the Ajax tool tip*

Summary

Well, how was that for a crash course on some rather complicated, but basic client/server Ajax/PHP examples? You have combined extensive knowledge in JavaScript, DHTML, Ajax, and PHP to create a very cool set of little applications. Considering that you've only scratched the surface, imagine all of the good stuff you can come up with when you start getting into the more advanced topics!

For now, it is merely important to see the basics behind using Ajax as a concept. First off, you should note that you will be doing far more programming in JavaScript than you may be used to. For me, when I first started working with Ajax, I found this to be a rather complicated issue—but I can assure you that your JavaScript skills will improve with time. It is imperative that you become good with CSS and such helpful tools as Firefox's JavaScript console and its DOM inspector. The JavaScript console (shown in Figure 3-5), in particular, is very efficient at pointing out any JavaScript syntax errors you may have accidentally put into place.

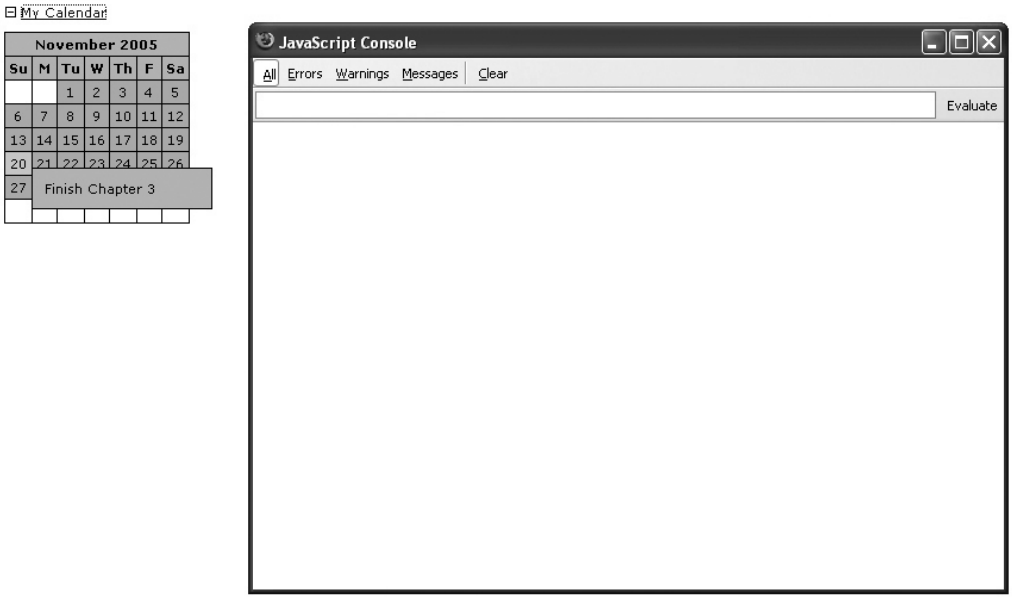

Figure 3-5. *The Firefox JavaScript console*

Once you have a firm grip on JavaScript and CSS, the possibilities for Ajax-based applications are endless. It is really a matter of getting the appropriate information to the appropriate PHP script, and then returning/displaying what you want. As you progress through the rest of this book, you will build upon the principles developed in this chapter to create some very powerful applications. For now, let's look at one of the more powerful aspects of server-side functionality: databases.