Apress™
Books for Professionals by Professionals

Chapter Fifteen: "JavaScript and COM"

A Programmer's Introduction to PHP 4.0

by William Jason Gilmore
ISBN # 1-893115-85-2

info@apress.com

CHAPTER 15

JavaScript and COM

As I've already stated several times throughout this book, one of the greatest aspects of the PHP language is the ease with which it can be integrated with other technologies. To this point, you've seen this in the discussions regarding databasing, ODBC, and XML, for example. In this chapter I introduce two more such possibilities for integration, namely, the ease with which PHP can work alongside JavaScript and COM-based applications.

Brief introductions to each of these technologies are provided in this chapter, in addition to practical examples of how PHP can interact with them. By the conclusion of this chapter, you will have gained useful knowledge about two particularly powerful technologies and how they can be further used through PHP.

JavaScript

JavaScript is a particularly powerful scripting language used to develop both client- and server-oriented Internet applications. One of the most interesting aspects of the language is the fact that it is capable of manipulating not only data but also events. An event is defined as any action that takes place in the realm of the browser, such as a mouse click or loading of a page.

For anyone who has experience working with programming languages such as PHP, C, Pascal, or C++, JavaScript will not be too much of a chore to learn. If you are a novice to these languages, do not be dismayed; JavaScript, like PHP, is a language that can be quickly learned. Like with PHP, its developers were content with creating a language geared toward one thing: getting the job done.

For those of you interested in retaining the event-driven capabilities of JavaScript while implementing the multitude of advantages offered by PHP, have no fear; PHP is as easily integrable with JavaScript as it is with HTML. In fact, it complements PHP unusually well, as it is capable of performing functions that PHP is not well suited for, and vice versa.

However, before attempting to integrate PHP and JavaScript, keep in mind that some users may have JavaScript turned off in their browsers or (gasp!) use a browser that does not even support JavaScript. PHP offers a simple solution for making this judgment.

JavaScript Detection

Correctly determining a browser's capabilities is essential to providing users a hassle-free visit to your site. After all, what could cause a user to leave a site faster than when annoying "JavaScript error" messages begin jumping out at them, or if the site functionality does not work properly due to the fact that the user's browser does not even support the technologies you are using? Fortunately, PHP offers an easy way to discern many of the capabilities supported by the user's browser, by way of a predefined function named `get_browser()`.

get_browser()

The function `get_browser()` retrieves browser capabilities, returning them in object format. The syntax is:

```
object get_browser([string user_agent])
```

The optional input parameter `user_agent` can be used to retrieve characteristics about a particular browser. Generally you will probably want to call `get_browser()` without any input parameter, since by default it uses the PHP global variable `$HTTP_USER_AGENT` as input.

A predefined list of browser capabilities is stored in the browscap file, its path directory specified by the `browscap` parameter in the php.ini file. The default setting of this parameter is shown here:

```
;browscap = extra/browscap.ini
```

The browser.ini file is developed by cyScape, Inc. The most recent version can be downloaded from http://www.cyscape.com/browscap/. Download and unzip the file, storing it in some logical location on the server. Take note of this location, as you will need to update the browscap setting in the php.ini file to reflect this location.

Once you have downloaded browscap.ini and configured your php.ini file, you are ready to begin incorporating browser-detection capabilities into your code. However, before doing so I would suggest opening the browser.ini file and scanning through it to get a better idea of how it is structured. After doing this, take a moment to experiment with Listings 15-1 and 15-2. Listing 15-1 is a very simple example that shows how all of a browser's characteristics can be output to the browser. Listing 15-2 focuses on the detection of just one of the browser's characteristics, JavaScript.

Listing 15-1: Showing all browser attributes

```
<?
// retrieve browser information
$browser = get_browser();

// typeset $browser to an array
$browser = (array) $browser;

while (list ($key, $value) = each ($browser)) :

     // clarify which of the browser elements are empty
     if ($value == "") :
          $value = 0;
     endif;

     print "$key : $value <br>";

endwhile;
?>
```

Executing Listing 15-1 in the Microsoft 5.0 browser yields the following output:

```
browser_name_pattern : Mozilla/4\.0 (compatible; MSIE 5\..*)
parent : IE 5.0
browser : IE
version : 5.0
majorver : #5
minorver : #0
frames : 1
tables : 1
cookies : 1
backgroundsounds : 1
vbscript : 1
javascript : 1
javaapplets : 1
activexcontrols : 1
win16 : 0
beta : 0
ak : 0
sk : 0
aol : 0
crawler : 0
cdf : 1
```

Listing 15-2 is a very short yet effective script that uses the browscap.ini file to verify whether or not JavaScript is enabled for a particular browser.

Listing 15-2: Ensuring JavaScript availability

```
<?
$browser = get_browser();

// typeset $browser to an array
$browser = (array) $browser;

if ($browser["javascript"] == 1) :
     print "Javascript enabled!";
else :
     print "No javascript allowed!";
endif;
?>
```

Basically, Listing 15-2 checks to verify whether the JavaScript key is listed for the given browser. If it is and is set to 1, then a message stating that JavaScript is enabled is displayed to the browser. Otherwise, an error message is displayed. Of course, in a practical situation you would likely get rid of the messages and perform other functions better suiting the user's browser capabilities.

The following two examples show just how easily PHP can be integrated with JavaScript. Listing 15-3 illustrates how the screen resolution and color depth can be determined using JavaScript and then subsequently displayed using PHP. The listing in the next section, "Building a Dynamic Pop-Up Window," illustrates how a PHP template can be used in a JavaScript-initiated pop-up window to display information based on the link the user clicks.

Listing 15-3: Detecting Color and Screen Resolution

```
<html>
<head>
<title>Browser Information</title>
</head>
<body>
<script language="Javascript1.2">
<!–//
document.write('<form method=POST action ="<? echo $PHP_SELF; ?>">');
document.write('<input type=hidden name=version value=' + navigator.appVersion +
'>');
document.write('<input type=hidden name=type value=' + navigator.appName + '>');
document.write('<input type=hidden name=screenWidth value=' + screen.width + '>');
```

```
document.write('<input type=hidden name=screenHeight value=' + screen.height +
'>');
document.write('<input type=hidden name=browserHeight value=' + window.innerWidth
+ '>');
document.write('<input type=hidden name=browserWidth value=' + window.innerHeight
+ '>');
//–>
</script>
<input type="submit" value="Get browser information"><p>
</form>

<?
     echo "<b>Browser:</b> $type Version: $version<br>";
     echo "<b>Screen Resolution:</b> $screenWidth x $screenHeight pixels.<br>";
     if ($browserWidth != 0) :
          echo "<b>Browser resolution:</b> $browserWidth x $browserHeight
          pixels.";
     else :
          echo "No JavaScript browser resolution support for this browser!";
     endif;
?>
</body>
</html>
```

Building a Dynamic Pop-Up Window

One of the interesting capabilities of JavaScript is the ease with which it can be used to manipulate browser windows. A useful application of this feature is small pop-up windows to display various parcels of information that perhaps are brief enough that they do not warrant the time taken to request and render another full page. Of course, you may be interested in creating a single template that will be used for each parcel of information. And thus the need for PHP. Listing 15-3 illustrates how a PHP file, window.php, is called from JavaScript. This file provides a very simple template, also incorporating a PHP INCLUDE call to the file ID as passed to window.php by the JavaScript in Listing 15-4.

For those of you not familiar with JavaScript, I have included descriptive comments in the code. The important point to keep in mind is that the variable `winID` will be passed to the PHP script, window.php This variable is assigned in the actual *link*, in the body of the HTML. Clicking it will trigger the function `newWindow()`, specified in the JavaScript. To illustrate how this is done, consider this link:

```
<a href="#" onClick="newWindow(1);">Contact Us</a><br>
```

As you can see, I simply place a '#' in the href, since the link will be generated by the JavaScript onClick event handler. This event handler causes the function newWindow() to be called when the user clicks the link. Finally, notice that the input parameter for the function is 1. This is the identification number that the PHP script will use to display the corresponding contact information. You can use any number you please, as long as it correctly corresponds to the file that will be displayed in the PHP script.

Take a moment to review Listing 15-4. As a guide, I have also built three simple files that correspond to each of the links in Listing 15-4.

Listing 15-4: Building dynamic pop-up windows

```
<html>
<head>
<title>Listing 15-4</title>
<SCRIPT language="Javascript">
<!–
     // declare a new JavaScript variable
     var popWindow;
     // declare a new function, newWindow
     function newWindow(winID) {
          // declare variable winURL, setting it to the name of the PHP file
          // and accompanying data.
          var  winURL = "window.php?winID=" + winID;
          // If the popup window does not exist, or it is currently closed,
          // open it.
          if (! popWindow || popWindow.closed) {
               // open new window having width of 200 pixels, height of 300
               // pixels, positioned
               // 150 pixels left of the linking window, and 100 pixels from the
               // top of the linking window.
               popWindow = window.open(winURL, 'popWindow',
               'dependent,width=200,height=300,left=150,top=100');
               }
               // If the popup window is already open, make it active and update
               // its location to winURL.
          else {
               popWindow.focus();
               popWindow.location = winURL;
          }
     }
//–>
</SCRIPT>
```

```
</head>
<body bgcolor="#ffffff" text="#000000" link="#808040" vlink="#808040"
alink="#808040">
<a href="#" onClick="newWindow(1);">Contact Us</a><br>
<a href="#" onClick="newWindow(2);">Driving Directions</a><br>
<a href="#" onClick="newWindow(3);">Weather Report</a><br>
</body>
</html>
```

Again, once the user has clicked one of the links in Listing 15-4, a pop-up window is created and window.php is displayed in this window. The variable `winID` is passed to window.php and is in turn used to identify the file that should be included in the PHP script. Listing 15-5 contains window.php:

Listing 15-5: window.php

```
<html>
<head>
<title>Popup Window Fun</title>
</head>

<body bgcolor="#ffffff" text="#000000" link="black" vlink="gray" alink="#808040"
marginheight="0" marginwidth="0" topmargin="0" leftmargin="0">

     <table width="100%" border="0" cellpadding="0" cellspacing="0">
          <tr>
               <td>
               <?
               // Include file specified by input parameter
               INCLUDE("$winID.inc");
               ?>
               </td>
          </tr>
          <tr>
               <td>
               <a href="#" onClick="parent.self.close();">close window</a>
               </td>
          </tr>
     </table>

</body>
</html>
```

The final piece to this puzzle is the creation of the files that correspond to the links in Listing 15-4. Since there are three unique IDs (1, 2, and 3), I need to create three separate files. These files are shown here. The first file, which holds the contact information, should be saved as 1.inc.

```
<table>
     <tr>
          <td>
          <h4>Contact Us</h4>
          <ul>
          <li>email: <a href="mailto:wj@wjgilmore.com">wj@wjgilmore.com</a>
          <li>phone: (555) 867 5309
          <li>mobile: (555) 555 5555
          </ul>
          </td>
     </tr>
</table>
```

The next file, which holds the driving directions, should be saved as 2.inc.

```
<table>
     <tr>
          <td>
          <h4>Driving Directions</h4>
          <ol>
          <li>Turn left on 1st avenue.
          <li>Enter the old Grant building.
          <li>Take elevator to 4th floor.
          <li>We're in room 444.
          </ol>
          </td>
     </tr>
</table>
```

And the final file, which holds the weather report, should be saved as 3.inc. To further illustrate the easy integration of PHP and JavaScript, notice how I make use of a call to PHP's date function:

```
<table>
     <tr>
          <td>
          <h4>Weather Report <?=date("m-d-Y");?></h4>
          <b>Today:</b> Brrr... Brisk, with blowing and drifting snow.<br><br>
          <b>Tonight:</b> Winter Weather Advisory. 7-10 inches snow expected.
          </td>
     </tr>
</table>
```

An example of what a pop-up window would look like if the user clicked the weather report is shown in Figure 15-1.

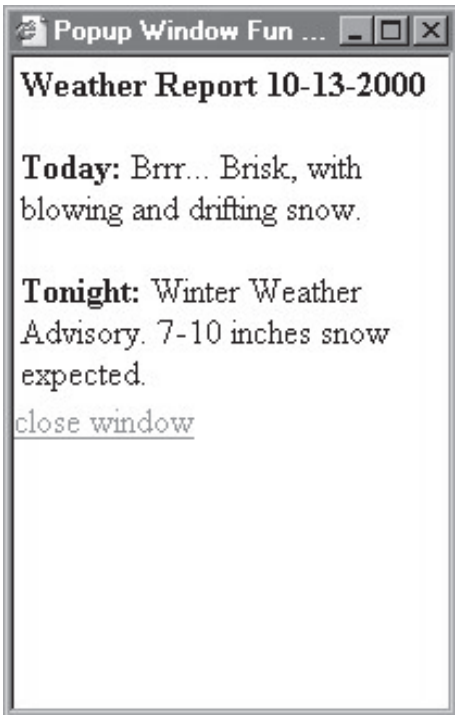

Figure 15-1. Displaying weather information in a pop-up window

And thus ends this ever-so-brief introduction to PHP and JavaScript integration. Also both examples are relatively simple, and both are useful and can be easily built on to suit more complex needs. Perhaps most important when combining PHP with JavaScript, or any other server-side-oriented technology, is that you must provide adequate means for detecting the capabilities of the user's browser

so as not to cause any ugly errors. In summary, it is always a good idea to experiment with other technologies in an attempt to incorporate them into your PHP code; just take heed so as not to scare the user away from your site due to unusable features or nonviewable content.

The next section discusses COM, another technology that can be easily interfaced using PHP.

The Component Object Model

COM, an acronym for *Component Object Model*, is essentially a specification that makes it possible for language- and platform-specific applications to communicate with each other. This capability greatly promotes the idea of building reusable, maintainable, and adaptable programming components, three ideas widely revered in the field of computer science. Although COM support is generally regarded as a Microsoft-centric specification, COM communication capability has actually been built into a number of languages (PHP, Java, C++, and Delphi, for example) and is used on a wide variety of platforms (Windows, Linux, and Macintosh, for example).

So what can COM and PHP do for you? Well, for one thing PHP's COM functionality makes it possible to directly communicate with many Microsoft applications. One interesting application is the formatting and display of Web database information to a Microsoft Word document. In fact, I'll show you just how easily this is accomplished in a later section.

> **NOTE** *These few pages devoted to COM barely, and I mean barely, scratch the surface of the technology. To make matters worse, it is also largely underdocumented as it relates to the PHP language. Therefore, if you are interested in learning more about the mechanics of COM, I would suggest checking out "Further Reading" at the end of this section.*

PHP has several predefined COM functions. Keep in mind that these functions are *only* available for the Windows version of PHP! Before moving on to some concrete examples of how they are used, please take a moment to review each as they are introduced below.

PHP's COM Functionality

PHP's predefined COM functionality is used to instantiate COM objects and subsequently make use of the objects' properties and methods. Remember that this support is only offered in the Windows version of PHP.

The accompanying examples will be based on working with Microsoft Word 2000. The objects, methods, and events referenced can be found at the Microsoft MSDN Web site (http://msdn.microsoft.com/library/officedev/off2000/wotocobjectmodelapplication.htm).

Instantiating a COM Object

To instantiate a COM object, just call `new`, just like you do to instantiate an object when programming the object-oriented way. The syntax is:

```
object new COM("object.class" [, string remote_address])
```

The parameter `object.class` refers to some COM module present on the server machine. The optional input parameter `remote_address` is used if you would like to create a COM object on some remote machine. For example, suppose that you want to instantiate an object that points to the MS Word application. This will actually start the Microsoft Word application just as if you had done so manually (of course, you must have MS Word installed on the machine in order for it to open). The syntax for doing so is shown here:

```
$word=new COM("word.application") or die("Couldn't start Word!");
```

Once you have instantiated a new COM object, you can begin working with the various methods and properties comprising that object. In regard to the above example, you may want to make the Word interface the active window. The following line enables the object's visibility attribute, resulting in the display of the application's interface:

```
$word->visible = 1;
```

Don't worry if you don't completely understand this command. Implementation of COM object methods is the subject of the next section.

Implementing a COM Object's Methods

You implement a COM object's methods in typical OOP format, using the object as a referring variable. The syntax for doing so is:

```
object->method_name([method_value, …]);
```

The `object` refers to a COM object that has been instantiated using the new instantiation process described previously. The parameter `method_name` refers to a method that is part of the class represented by `object`. The optional parameter space specified by `method_value` can be used to input parameters to those methods that allow or require input. Its syntax is just like that of a normal function, with each input parameter separated by a comma.

If you wanted to open a new MS Word document after instantiating a new COM object pointing to the application, as seen in the previous example, you could simply reference the method that accomplishes this. This is the add() method in the Documents subclass of `$word`:

```
$word->Documents->Add();
```

Notice how this follows a very logical, OOP-style syntax. Executing this will result in a new document being displayed to the MS Word application window.

com_get()

The function `com_get()` is used to retrieve COM object properties. Its syntax is:

```
mixed com_get(resource object, string property)
```

The input parameter `object` points to an instantiated COM object, and `property` refers to an attribute in the class represented by the instantiated object.

```
<?
// Instantiate a new object pointing to the MS Word application
$word=new COM("word.application") or die("Couldn't start Word!");

// The CapsLock property is either 0 for No, or 1 for Yes.
$flag = com_get($word->Application,CapsLock);

// Turn $flag value (0 or 1) into human-readable format
if ($flag == 1) :
     $flag = "YES";
else :
     $flag = "NO";
endif;
// display appropriate message
print "CAPS Lock activated: $flag";
$word->Quit();
?>
```

Alternatively, you could retrieve the `CapsLock` attribute value by calling it just as you would call an object's attribute via OOP syntax. To use this alternative format in the above example, simply replace this line in the above example:

```
$flag = com_get($word->Application,CapsLock);
```

with this line:

```
$flag = $word->Application->CapsLock;
```

Making use of these object attributes, you can retrieve any variety of information about the characteristics of an application. Furthermore, you can also set values for many characteristics. This is accomplished with the function `com_set()`.

com_set()

The function `com_set()` is used to set an object attribute to a specified value.

```
void com_set(resource object, string property, mixed value)
```

The input parameter `object` points to an instantiated COM object, and `property` refers to an attribute in the class represented by the instantiated object. The parameter `value` is the value to which you would like to set `property`.

In Listing 15-6, the Microsoft Word application is started and made the active window. A new document is then created, and one line of text is added to it. Next, I set the default document format (the attribute is called `DefaultSaveFormat`) to Text. This will become apparent once the Save As prompt is displayed, as you will see that the Save As Type setting is set to Text Only. Once you save the document, the Microsoft Word application is closed.

Listing 15-6: Setting the default document type

```
<?
// Instantiate a new object pointing to the MS Word application
$word=new COM("word.application") or die("Couldn't start Word!");

// Make MS Word the active window.
$word->visible =1;

// Create a new document
$word->Documents->Add();

// Insert some text into the document
```

```
$word->Selection->Typetext("php's com functionality is cool\n");

// Set the default document format to Text
$ok = com_set($word->Application, DefaultSaveFormat, "Text");

// Prompt the user to name and save the document.
// Notice that the default document format is Text!
$word->Documents[1]->Save;

// Quit MS Word
$word->Quit();
?>
```

Alternatively, you could set the `DefaultSaveFormat` attribute by directly calling it almost like you would a variable. To achieve the same results as Listing 15-6 using this alternative format, simply replace the line:

```
$ok = com_set($word->Application, DefaultSaveFormat, "Text");
```

with the line:

```
$word->Application->DefaultSaveFormat = "Text";
```

At this point, you have been introduced to all of the functionality necessary to manipulate Windows applications via PHP's COM functionality. Now I'll move on to a rather interesting example that illustrates just how useful and cool the COM features can be.

Writing Information to a Microsoft Word Document

This example demonstrates just how useful PHP's COM functionality can be. Suppose some of your users wanted to format some database information in a Microsoft Word document for a presentation. Just a few lines of PHP code can automate this entire process. To illustrate this, I'll use the table addressbook first used in the address book project at the end of Chapter 12. The process executed by the script flows as follows:

1. Connect to the MySQL server and select the necessary database.

2. Select all of the data in the table, ordering it by last name.

3. Open the Microsoft Word application and create a new document.

4. Format and output each row of table data to this document.
5. Prompt the user for a name under which the document will be saved.
6. Close Microsoft Word

The code is shown in Listing 15-7.

Listing 15-7: Interacting with Microsoft Word through PHP's COM functionality

```
<?
// Connect to the MySQL server
$host = "localhost";
$user = "root";
$pswd = "";
$db = "book";
$address_table = "addressbook";

mysql_connect($host, $user, $pswd) or die("Couldn't connect to MySQL server!");
mysql_select_db($db) or die("Couldn't select database!");

// Query the company database for all 'addresses' rows
$query = "SELECT * FROM $address_table ORDER BY last_name";
$result = mysql_query($query);

// Instantiate a new COM object. In this case, one pointing to the MS Word
application
$word=new COM("word.application") or die("Couldn't start Word!");

// Make MS Word the active Window
$word->visible =1;

// Declare a new, empty document.
$word->Documents->Add();

// Cycle through each address table row.
while($row = mysql_fetch_array($result)) :
     $last_name = $row["last_name"];
     $first_name = $row["first_name"];
     $tel = $row["tel"];
     $email = $row["email"];

     // Output table data to the open Word document.
```

```
        $word->Selection->Typetext("$last_name, $first_name\n");
        $word->Selection->Typetext("tel. $tel\n");
        $word->Selection->Typetext("email. $email:\n");

endwhile;

// Prompt the user for a document name
$word->Documents[1]->Save;

// Quit the MS Word Application
$word->Quit();
?>
```

Although this example is very simple, it illustrates in a very practical sense how you could write PHP applications that synchronize database information with a user's favorite Windows application. A more complicated application could be written that would allow users to sync Web-viewable information with Microsoft Outlook. All you would need to do is obtain a reference of Outlook's objects, properties, and methods, and you can begin experimentation. (An introduction to the object model of all applications comprising Office is at `http://www.microsoft.com/officedev/articles/Opg/toc/PGTOC.htm`).

Further Reading

The following links point to several of the more useful COM-related resources that I have found on the Internet:

- `http://msdn.microsoft.com/library/techart/msdn_comppr.htm`
- `http://www.microsoft.com/Com/news/drgui.asp`
- `http://www.microsoft.com/com/default.asp`
- `http://www.comdeveloper.com/`

What's Next

This chapter further introduced just how easy it is to integrate PHP with third-party technologies, namely, JavaScript and the Component Object Model (COM). In particular, I introduced the following topics:

- What is JavaScript?
- Detecting JavaScript-capable browsers
- Detecting browser properties
- Using pop-up windows in conjunction with PHP
- What is the Component Object Model (COM)?
- PHP's predefined COM functionality
- Using PHP's COM functionality to send database data to Microsoft Word

Integrating these technologies with PHP can expand the functionality of your applications in many ways. Working with JavaScript opens up the possibility of performing certain functions on the client side, such as window and browser manipulation and forms error checking. COM provides you with the possibility to create applications that communicate directly with such popular applications as the Microsoft Office suite, further enhancing the value and user-friendliness of your PHP applications.

In our final chapter (Did it really go by that quickly?), I cover a topic that should be constantly on the minds of every programmer and administrator: security. Important security-related issues such as script protection, encryption, and ecommerce data solutions are introduced.