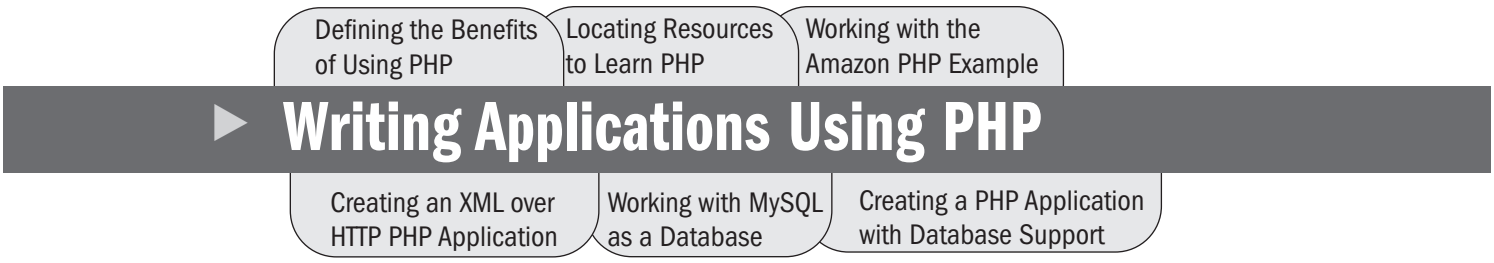
Chapter 7



Many developers have learned to use PHP over the years because it's a good solution for creating Web pages and the price is right. The PHP acronym is like many other new acronyms for the Internet—the acronym is recursive (refers back to itself). PHP stands for *PHP Hypertext Processor*. This general-purpose HTML scripting language works much like ASP (see Chapter 6) or other page description languages you might have used. Essentially, you mix HTML with scripting information. When the PHP process sees HTML, it sends the text directly to the user. It processes any scripting information, and passes the resulting HTML to the user as well.

This chapter helps you discover how PHP works with Amazon Web Services. I'm assuming that you already know something about PHP, but the first two sections provide some suggestions on how to learn more about PHP. Because PHP runs on so many platforms, you'll also find some suggested resources for getting and installing PHP for your particular server. These instructions might require a little technical knowledge on your part, so make sure you understand what the instructions require before you perform them.

The examples in this chapter show how to use PHP to create an Amazon Web Services application. The first example provides simple instructions for accessing the Web service without any fancy application features. You'll also find an application that shows how to use PHP with MySQL, an open source database. In fact, you can download every piece of software in this chapter free and try out all of the examples without spending a penny—that's one of the benefits of using open source.

I've also provided a number of tips to help you with your PHP applications. You'll find that the open source support system is adequate, but you won't get the same level of hand-holding that you do with paid products such as Visual Studio. Open source solutions tend to require a motivated developer, so it might not be the right solution if you need a packaged approach that doesn't require a lot of fiddling on your part. With this in mind, the chapter also provides some ideas on where you can get help when you need it.

Understanding the Benefits of Using PHP

PHP has a number of interesting benefits, especially for a company on a budget. One of the biggest benefits is that PHP is essentially free because it's open source, as is the main Web server it runs on—Apache. All you need to do is download the required products and install them on your system. In fact, you'll find an amazing array of products you can use with Apache and PHP on the Apache Software Foundation site at `http://www.apache.org/index2.html`. Note that this site also keeps you informed about many of the conferences associated with the open source movement and many of the political issues as well.

Another important benefit of using PHP is that it runs just about anywhere. The Apache server comes in versions for Windows, NetWare, Linux, Macintosh OS/X, and most Unix systems. You can also find Apache support for larger systems such as the AS/400. In fact, there are few places that Apache doesn't run. Anywhere you can run Apache, you can likely run PHP. In the few cases you can't find a version of Apache to use for your copy of PHP, it's quite possible you can find a version of PHP that runs on another Web server for that platform.

Understanding the Usage of *Free* with Software

Free can be a subjective term. Free doesn't necessarily mean without cost. In many cases, you see the word *free*, but it doesn't mean that everything about a product is free. The problem is that I haven't come up with better terminology. The use of the term free has caused so much worry for users that some of them have sued vendors over the use of the term free because it really doesn't mean inclusively free for the entire product. This article sums up the situation: `http://www.infoworld.com/article/03/08/29/34FElinux_1.html`.

As you can see from this (and other) articles, the experts haven't yet placed a price tag on the supposedly free Linux. Yes, you can download a generic copy of Linux free, but most people choose a customized version, so that costs money, but it costs less than a copy of Windows. Unfortunately, the cost of the operating system is a relatively small percentage of the whole cost of installing a system. Even if a company chooses the free generic version of Linux, no one can say that it's truly free, but few experts agree on the actual cost.The political part of the equation comes from the uncertainty of cost. If you're Microsoft and you want to dissuade someone from buying Linux, then you make those costs as high as you can without losing support from the "independent" experts who will back up your claim. On the other hand, if you're a member of the open source community, you want to make those costs as low as possible without calling your sanity into question.

This chapter won't discuss the political issues that surround the concept of *free*. For the purposes of this book, free means the product won't cost anything to download. You still have to consider support, installation, and management costs as part of any solution you use.

PHP is relatively easy to transport from one system to another. Because PHP applications run as scripts (essentially text), any application you create for PHP on one platform is likely to work with a few tweaks on another platform. Amazon Web Services developers should find that PHP works especially well for multiple platforms. Problems can occur when you begin adding platform-specific features to an application to make it look nicer or perform better.

The creators of PHP have improved it a great deal since its initial release. For example, you no longer need Apache to run PHP (many Web sites and articles still say this is a requirement). See the "Downloading and Installing PHP" section for additional information on this topic. The open source community also provides regular patches for PHP, including the all important security patches. You can find these patches on the PHP download site. Because these patches receive an open review, many developers consider them better coded and more stable than the proprietary solutions available on the market.

▶ NOTE

I'm using the Apache 2.0.47 Windows and PHP 4.3.3 versions for this chapter. You might notice some differences between these product versions and other versions available on the download sites. Because of the way PHP works, the example code should work fine on any newer version of Apache and PHP you choose to use. Older versions of both Apache and PHP could encounter problems when they don't support the features found in the current products.

Resources for Learning PHP

This chapter assumes that you already know how to use PHP and simply want to learn how to use it with Amazon Web Services. Consequently, the chapter doesn't include essential language instruction that you might need if you're a PHP novice. If you think you might want to learn to use PHP for your next Web application, the resources in this section will help. One of the first places you should look for PHP information is the PHP site at `http://us2.php.net/manual/en/introduction.php`. Once you spend some time with the PHP tutorial, you'll also want to look at the PHP manual at `http://us2.php.net/manual/en/index.php`. The manual tells you how to use various PHP commands.

The Webmonkey Web site has an excellent PHP tutorial (`http://hotwired.lycos.com/webmonkey/01/48/index2a.html?tw=programming`). Another tutorial will help you understand PHP and MySQL Usage (`http://hotwired.lycos.com/webmonkey/programming/php/tutorials/tutorial4.html`). However, you'll also want to view the other PHP

topics on this Web site (`http://hotwired.lycos.com/webmonkey/programming/php/index.html`) to learn more about PHP and see how you can use it with other products such as Oracle.

A number of other sites also provide PHP tutorials. For example, the Free Webmaster Help.com site at `http://www.freewebmasterhelp.com/tutorials/php` provides a seven-part tutorial that includes information on using forms. The tutorials on Dev Shed (`http://www.devshed.com/Server_Side/PHP/`) are a little more advanced. The tutorials on this site help you discover how to work with the local hard drive and even create PDFs as output from your application. You'll also find a number of articles about error handling and other developer topics. However, you'll want to save this site as your last stop because many of the articles get quite detailed and you could find yourself lost quickly.

It's also helpful to have a good book on the topic. Take a look at *Creating Interactive Web Sites with PHP and Web Services* by Eric Rosebrock (Sybex, 2004). This book shows how to install and configure development and production platforms of Apache, PHP, and MySQL on both Windows and Linux systems, and teaches Web development with PHP from a problem-solving viewpoint. Also visit Eric's wildly popular Web site PHP Freaks (`http://www.phpfreaks.com`).

Downloading and Installing PHP

One of the first places you'll want to visit is the Webmonkey site at `http://hotwired.lycos.com/webmonkey/00/44/index4a.html?tw=programming`. Use this tutorial to get PHP setup on your system and learn a little about this product. This PHP tutorial will also introduce you to the language. Unfortunately, the tutorial is also a little outdated and many of the links no longer work. Here's a list of links you can use instead of the links provided with the Webmonkey article (the article information is still very good, so don't be concerned about the outdated links).

- Apache Server Download (`http://httpd.apache.org/download.cgi`)
- Apache Documentation (`http://httpd.apache.org/docs-2.0/`)
- PHP Download (`http://us2.php.net/downloads.php`)
- PHP Manual (`http://us2.php.net/manual/en/index.php`)

Because the Webmonkey article is a little outdated, you'll also want to spend time with the official PHP installation documentation found at `http://us2.php.net/manual/en/installation.php`. Although this text isn't quite as readable as the Webmonkey version, it's definitely current. Make sure you base any installation decisions, such as whether to use CGI or ISAPI, on the content of the official documentation.

One thing you won't need to do to work with Amazon Web Services is add any extensions. All of the examples in this chapter work fine with the default extensions. You might need to add extensions to process the data, but it's a good idea to work with Amazon Web Services for a while using the default PHP configuration. Using the default configuration ensures you won't run into any extension-specific errors.

Don't get the idea that PHP only comes in versions for Apache users. It's true that many people use PHP with Apache, but you can also use it with Internet Information Server (IIS), Personal Web Server (PWS), and Xitami (among other servers). Many of the other servers require that you use the CGI version of PHP, but you can also get an ISAPI version for IIS. The ISAPI version will provide superior performance and a little more flexibility, as well as improved reliability and recoverability. If you want the ISAPI support, you must download the Zip version of the PHP file, not the installer version, which includes only the CGI files.

You can run into a number of issues with Apache that none of the documentation mentions. For example, you might run into a situation where Apache installs and even starts, but you can't access it. Make sure you don't have another Web server installed on the same system. The second Web server could make it difficult or impossible to access the Apache server. This problem is especially prominent on Windows systems because Microsoft simply assumes that every server should have IIS installed.

Amazon Web Services can accept either XML over HTTP or SOAP requests from PHP. The choice of interface depends on what you plan to achieve with the Web service. In many cases, you can achieve acceptable results using the XML over HTTP approach with less coding and effort than using the SOAP approach. If you decide to use the SOAP approach, you'll need to download a SOAP library to use with PHP. You can find this

▶ TIP

Like most programming languages, you'll find a variety of third party support sites for PHP. One of the better sites, ByKeyword.com (`http://www.bykeyword.com/pages/php.html`) includes a list of utilities to edit, manage, and even convert your PHP code. Make sure you also visit sites like Tucows (`http://tdconline.tucows.com/`). A simple search can net a list of useful shareware and freeware products you can use.

▶ NOTE

At the time of this writing, Webmasters have closed some PHP sites temporarily as a protest against the legal wrangling occurring in the open source community. For example, the php4win site (`http://www.php4win.de/`) won't allow any access at all. Consequently, if you see that one of the sites mentioned in this chapter is closed, try it again after a month or so.

library at http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/nusoap/lib/nusoap.php. This file must appear in a central location or in the same folder as your other application files.

Using the Amazon Supplied File Example

The Amazon Web Services Kit includes a PHP example you can use for experimentation. This example is useful in several ways. Obviously, it shows how to develop a PHP application. However, the example includes a second file, AmazonSearch.PHP, which greatly simplifies your SOAP programming needs if the functions it includes meets your requirements. This section discusses both uses of the example.

First, you need to make a change to the source code for the Amazon.PHP file. The example won't work properly as a Web page as written now. Open the file in any plain text editor. You can use Notepad, but an editor designed for the task works better. I wrote all of the examples in this chapter using PHP Expert Editor (http://www.phpexperteditor.com/). You can find a list of additional editors at http://phpeditors.linuxbackup.co.uk/editorlist.php. At line 58 of the Amazon.PHP file, you'll find a line that reads

```
if (count($argv) == 1)
```

Change this line so it reads

```
if (count($argv) <= 1)
```

Now that you have the code fixed, copy the entire \PHPSample folder to the \Program Files\Apache Group\Apache2\htdocs folder of your server. In addition, copy the NuSOAP.PHP file to this folder. You should find four files in the \Program Files\Apache Group\Apache2\htdocs\PHPSample folder at this point. Make sure the server has a connection to the Internet and access to the example. Figure 7.1 shows typical output from the example. Note that you can obtain a better view of the data by right-clicking the browser window and selecting View Source from the context menu.

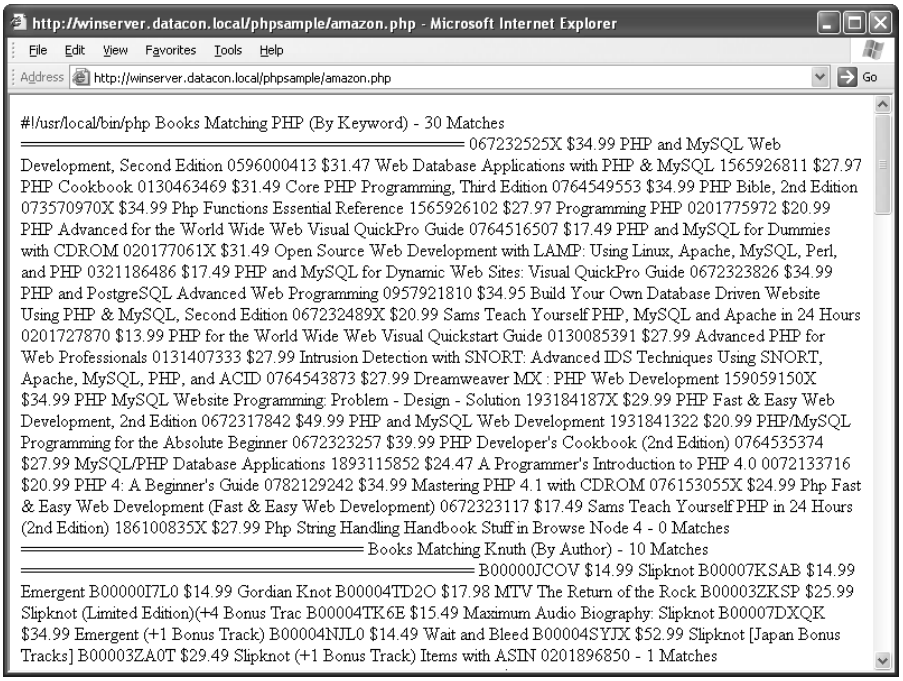
While the example is interesting, it's not the complete picture. The AmazonSearch.PHP file is the one that contains the code that makes everything happen. Amazon wisely placed this code in a separate file so you could easily use the functions it contains in your own applications. Using this file makes the SOAP technique almost too simple because all you need to know is which function to use and how to parse the incoming data.

▶ NOTE

The Amazon developer originally intended this application to run at a command prompt as a script, not as a Web page. The code change allows you to run the code as either a script or a Web page. However, the arguments mentioned at the top of the file only work when you use the example at the command line. To use the code this way, you would type **PHP Amazon.PHP -a** at the command prompt and press Enter to obtain the author query results. Make sure you include any required path information to locate the PHP.EXE file.

Most applications you create using the Amazon supplied files require three steps. The first step is to create a request and retrieve results from that request. Listing 7.1 shows an example of how you can use the `AmazonSearch.PHP` file to perform this first step. You'll find this example in the `\Chapter 07\AmazonSearch` folder of the source code located on the Sybex Web site.

FIGURE 7.1: The Amazon PHP example shows some of the results you can achieve.



Listing 7.1 Building an Application Using the Amazon Supplied Files

```
<?php
// Set up the variables used to make the request.
$Debug   = false;   // Use debug mode?

// Associate token.
if ($_REQUEST["txtTag"] == null)
   $Tag    = "webservices-20";
else
   $Tag    = $_REQUEST["txtTag"];

... Other Inputs ...

// Number of return results.
if ($_REQUEST["txtNumber"] == null)
   $Return = 10;
```

```
else
   $Return = $_REQUEST["txtNumber"];

// Load Amazon Search object
require("./AmazonSearch.php");

//  Initialize the search service.
$Service = new AmazonSearch($DevTag, $Tag, $Debug);

//  Perform an Author search.
$Results = $Service->DoAuthorSearch($Author, $Type, $Mode, $Return);

?>
```

The code checks each input for a request value and uses a default if it doesn't find one. As in many cases, a user could open the page directly or as the result of clicking Submit on the form. You could simply display the form without making a request if the user hasn't pressed Submit.

Once the code has input values to use, it creates an instance of Amazon Web Services. This object is actually the class that Amazon supplies as part of the kit. You need to create all of the data that the class creates by hand if you decide that the predefined class won't do what you need. Notice that the constructor requires both the developer and associate tokens. You can also place the class in a debug mode by setting `$Debug` to true.

At this point, the code can make a query. The example uses the `DoAuthorSearch()` method, which requires the four inputs shown. One of the advantages of this method is that you can define the number of results you want returned from the query. Theoretically, you could make just two calls to Amazon Web Services to obtain all of the results for a particular request. The first call simply retrieves the number of results available, while the second call requests the full number of results. The only problem with this method is that you can't select a page. The method always begins with the first page of available results, which means every request returns the same starting information. It's possible to change the Amazon supplied code to add functionality such as the starting page and sort order, so you may still want to use this file as a starting point.

The second step of the process displays the HTML portion of the Web page. Listing 7.2 shows a typical example of this portion of the code.

Listing 7.2 Creating the User Display Using the Amazon Supplied Files

```
<html>

<head>
<title>Author Amazon Search Demonstration</title>
</head>
```

```
<body>

  <form action="http://YourWebsiteURL/AmazonSearchDemo.PHP"
        id="SubmissionForm"
        method=get
        name="SubmissionForm">
     <h1 align=center>
        <label id="Heading">Query Entry Form</label>
     </h1><p />
     <label id="Type">Associate
        <span style="TEXT-DECORATION: underline">T</span>ag:
     </label>
     <input id="t" accesskey="T" type="text"
        <?php
           if ($_REQUEST["txtTag"] == null)
              print('value="webservices-20"');
           else
           {
              print('value="');
              print($_REQUEST["txtTag"]);
              print('"');
           }
        ?>
        name="txtTag"><br/>

     ... Other Labels and Inputs ...

     <div align="center">
        <input id="Submit" type=submit value="Submit" NAME="Submit">
     </div>
  </form>

  <table align=center width=80% border=1>
     <caption>
        <span style="FONT-WEIGHT: bold; FONT-SIZE: large">
           Query Results
        </span>
     </caption>
     <tr>
        <th>ISBN</th>
        <th>Product Name</th>
        <th>Release Date</th>
        <th>Publisher</th>
     </tr>
```

For the most part, this code looks like a standard PHP page. It does include special code to display the results of any changes the user makes to the page and submits to the server. Because Amazon Web Services code relies on these changes, you must include code of this type to ensure the user changes remain in place. I also chose to place the output table outside the form to avoid problems with data submissions.

The third step of the process is to display the data retrieved with the Amazon request. Listing 7.3 shows one way to work with this data.

Listing 7.3 Processing the Results Using the Amazon Supplied Files

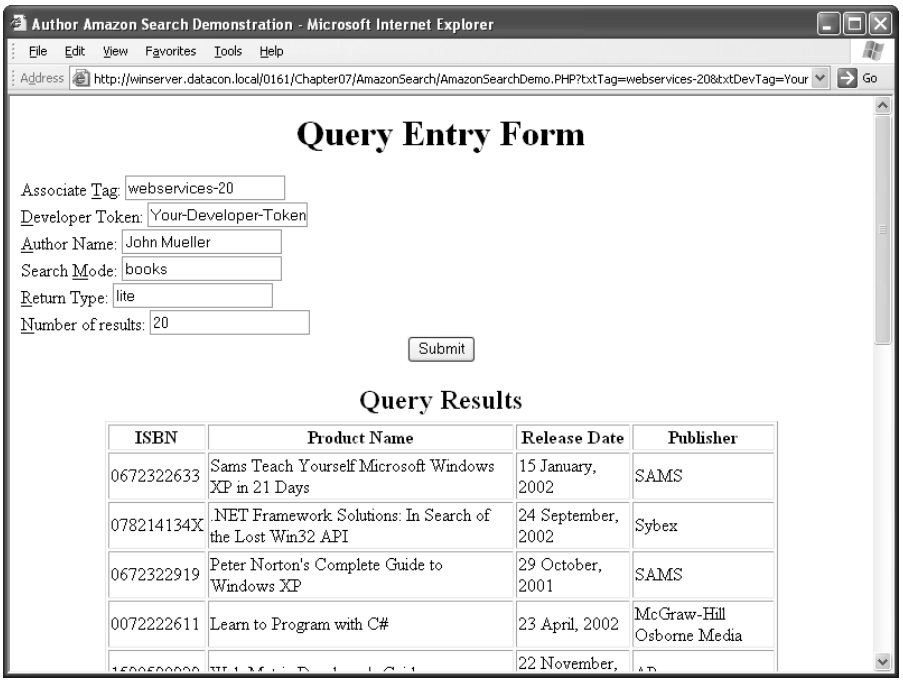
```
<?php
  foreach($Results as $Detail)
  {
     // Start a row.
     print("<tr>");
     print("\r\n");

     // Get the ISBN.
     print("<td>");
     print($Detail["Asin"]);
     print("</td>");
     print("\r\n");

     ... Other Results ...

     print("</tr>");
     print("\r\n");
  }
?>
```

As you can see, PHP provides a `foreach` statement that works well with the data because of the way it's organized. The `AmazonSearch` class treats the return value as an array—each array element contains one book detail record. This record is arranged the same as any Amazon `ProductInfo` data structure. Consequently, you can retrieve some values such as the ASIN directly, but other values, such as Authors, require further processing. Figure 7.2 shows typical output from this application.

Developing an XML over HTTP PHP Application

You might run into a situation where you want to use the XML over HTTP technique with PHP. Certainly, putting the message together is much easier than working with SOAP and you don't need to download anything extra—the standard PHP setup includes everything you need. Some people also find this solution a tad more flexible because you can perform tasks at the start of the element, during the reading of the character data, and again at the end of the element. The example in this section performs a simple query of Amazon. Processing the data requires two steps. The first step is to open the remote connection, get the data, and start the processing loop. This step appears in Listing 7.4. You'll find this example in the `\Chapter 07\XMLDemo` folder of the source code located on the Sybex Web site.

FIGURE 7.2:
Typical output from the sample application includes as many entries as needed.



Listing 7.4 Initiating the XML Processing Loop

```
// Define the XML parser and set the parsing functions.
$Output = xml_parser_create();

// Define the element parser.
xml_set_element_handler($Output, "StartElement", "EndElement");
xml_set_character_data_handler($Output, "CharacterData");

// Open a connection to Amazon using the XML over HTTP method.
$FilePointer = fopen("http://xml.amazon.com/onca/xml3?" .
                     "t=webservices-20&dev-t=Your-Developer-Token&" .
                     "AuthorSearch=John%20Mueller&mode=books" .
                     "&type=lite&page=1&f=xml", "r");
if ($FilePointer == null)
   // Display an error message an exit.
   die(print("Couldn't open the Amazon site."));
else
{

   // Keep reading the data until there isn't any more to read.
   while ($DataStream = fread($FilePointer, 4096))
```

```
    {
       // If the program can't parse the data, raise an
       // error.
       if (!xml_parse($Output, $DataStream, feof($FilePointer)))
       {
          // Display an error message and exit.
          die(sprintf("XML error: %s at line %d",
             xml_error_string(xml_get_error_code($Output)),
             xml_get_current_line_number($Output)));
       }
    }
}
```

The code begins by creating the XML parser, `Output`, using the `xml_parser_create()` function. It then assigns an element handler and a character data handler to the XML parser using the `xml_set_element_handler()` and `xml_set_character_data_handler()` functions. These three functions actually process the incoming data.

Now that the code has an XML parser to use, it opens a pointer to the XML data using the `fopen()` function with the same string as the example in the "Using a Browser Example" section of Chapter 2. If the `fopen()` function is successful, the code begins the processing loop.

Processing involves reading data from the data stream into `$DataStream` using the `fread()` function. The code parses the resulting data using the `xml_parse()` function. What actually happens in this loop is that the code calls the three functions found in Listing 7.5 in turn.

Listing 7.5 Outputting the Processed Data as HTML

```
$CurrentElement = "";

function StartElement($Parser, $Name, $Attributes)
{
   // Store the current element name.
   global $CurrentElement;
   $CurrentElement = $Name;

   // Start a table containing the arguments.
   if ($Name == "ARGS")
   {
      print('<table align=center width=50% border=1>');
      print('<caption>');
      print('<span style="FONT-WEIGHT: bold; FONT-SIZE: large">');
      print('Arguments</span>');
      print('</caption>');
      print('<tr>');
      print('<th>Name</th>');
```

```
      print('<th>Value</th>');
      print('</tr>');
   }

   // List each argument in turn.
   if ($Name == "ARG")
   {
      print('<tr><td>');
      print($Attributes["NAME"]);
      print('</td><td>');
      print($Attributes["VALUE"]);
      print('</td></tr>');
   }

   // End the arguments table and print the total results.
   if ($Name == "TOTALRESULTS")
   {
      print("</table>");
      print("<LABEL>Total Results: ");
   }

   // Print the total number of pages.
   if ($Name == "TOTALPAGES")
      print("<LABEL>Total Pages: ");

   // Print the ASIN.
   if ($Name == "ASIN")
      print("<tr><td>");

   ... Other Data Values ...

}

function EndElement($Parser, $Name)
{
   // End the Total Result value.
   if ($Name == "TOTALRESULTS")
      print("</LABEL><br/>");

   // End the Total Pages value. Also set the
   // page up to print the details table.
   if ($Name == "TOTALPAGES")
   {
      print("</LABEL>");
      print('<table align="center" border="1" width="90%">');
      print('<caption>');
      print('<span style="FONT-WEIGHT: bold; FONT-SIZE: large">');
      print('Books Returned from Query</span>');
      print('</caption>');
```

```
            print('<tbody>');
            print('<tr>');
            print('<th>ISBN</th>');
            print('<th>Book Title</th>');
            print('<th>Release Date</th>');
            print('<th>Publisher</th>');
            print('</tr>');
        }

        if ($Name == "ASIN")
            print("</td>");

        ... Other Data Values ...

    }

    function CharacterData($Parser, $Data)
    {
        // Get the current element name.
        global $CurrentElement;

        // Print the Total Results value.
        if ($CurrentElement == "TOTALRESULTS")
            print($Data);

        // Print the Total Pages value.
        if ($CurrentElement == "TOTALPAGES")
            print($Data);

        if ($CurrentElement == "ASIN")
            print($Data);

        ... Other Data Values ...

    }
```

You must think about the three functions in terms of how you want data to appear on screen. The easiest method is to process the data as it comes from Amazon, which is what this example does. The processing look also calls these three functions in turn, which means that it passes an `<Args>` element to the `StartElement()` function first, but then it processes all of the `<Arg>` child elements by calling both the `StartElement()` and `EndElement()` functions before it calls `EndElement()` for the `<Args>` element. It's handy to keep a browser display of the raw data similar to the one shown back in Figure 2.4 to help create your code.

In this case, the code begins by creating a table in the `StartElement()` function to display the arguments on screen. It then processes the `<Arg>` elements as a table. Notice how the

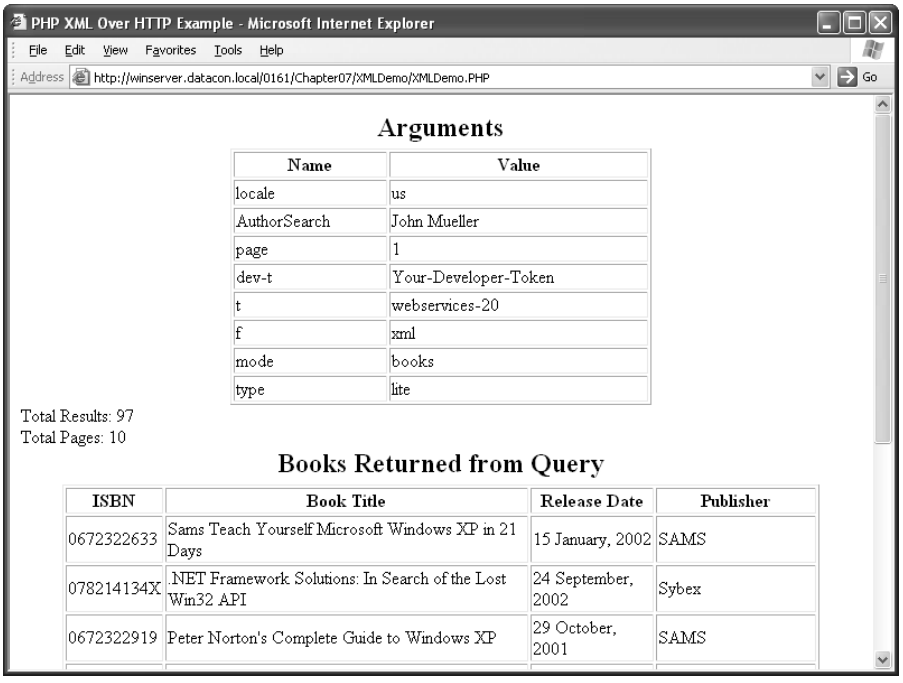
code uses strings to access the `$Attributes` array for the `<Arg>` elements. The `StartElement()` function continues by beginning to process the `<TotalResults>` and `<TotalPages>` elements. Element character data isn't accessible in this function, so the code only creates the beginning label. Likewise, since elements such as `<Asin>` use character data instead of attributes, the code only creates the required supporting tags.

Look now at the `CharacterData()` function. As you can see, it keeps track of the current element using the global variable, `$CurrentElement`. Whenever the code comes to an element that contains data that it should present on screen, it simply prints the `$Data` variable.

Now that the code has printed an opening tag and the required data, the Web page requires a closing tag. The `EndElement()` function takes care of this task. The tags that you need to output depends on what you want to do with that element, what you did in the `StartElement()` function, and what the next element will need to work right. In this case, most of the entries are obvious—they simply output the closing tag. However, the `<TotalPages>` element also marks the beginning of the details processing. Because the details aren't contained within a parent element, adding the table tags here makes sense. Figure 7.3 shows the output from this example.

FIGURE 7.3:

This example relies on XML over HTTP to provide detailed results.



Using MySQL as a Database

You'll probably want to improve the efficiency of your PHP application, at some point, by storing some data locally. It's likely that you'll use a database to perform this task. One of the more popular databases on the market is MySQL. It provides robust capability and the price is right. In addition, MySQL seems to enjoy better than average support from a cadre of developers who use it.

Like everything else in this chapter, MySQL is open source. Normally, you don't have to buy this product—just download it. However, there are some situations where you do need to buy a license, such as when you create an application for commercial (shrink-wrap) distribution where you'll realize a profit from the sale of the application. Make sure you understand the distribution requirements for MySQL by reading them at `http://www.mysql.com/downloads/index.html`.

The example in this chapter relies on MySQL 4.0, the latest production version at the time of writing, which you can download at `http://www.mysql.com/downloads/mysql-4.0.html`. You'll notice that MySQL comes in quite a few versions for various platforms including Linux, Windows, Solaris, FreeBSD, Mac OS X, HP-UX, IBM AIX, Novell NetWare, SCO OpenUnix, SGI Irix, and DEC OSF. You can also download the source code and create your own flavor of MySQL if necessary. The test system for this chapter uses the compiled Windows version with default settings applied by the installer. If you use some other form of MySQL, your screenshots will vary from mine.

Once you download the version of the product you need, install it according to the vendor directions. In most cases, this means starting the installer or unpacking the product and performing a manual install. The Windows Installer version is very easy to use—just double-click the executable that you download and follow the prompts.

Learning to use MySQL is relatively straightforward. You can find the complete product documentation at `http://www.mysql.com/documentation/index.html`. The documentation comes in two formats: PDF, for a printed version, or HLP, for a desktop electronic version. Part of the documentation is a tutorial that you'll find at `http://www.mysql.com/doc/en/Tutorial.html`. The vendor provides training and certification courses that you can learn about at `http://www.mysql.com/training-and-certification.html`. The Webmonkey site at `http://hotwired.lycos.com/webmonkey/programming/php/tutorials/tutorial4.html` provides an excellent online MySQL tutorial. Another good tutorial appears on the TAASC site at `http://www.analysisandsolutions.com/code/mybasic.htm`. In fact, you can find a number of tutorials on this product.

Writing a PHP Application with Database Support

The example in this chapter replicates most of the functionality of the example in the "Using SQL Server as a Database" section of Chapter 6. Of course, that application runs on a desktop machine and this one works as a browser application, so there are some differences. For one thing, the application stores the URL of the local copy of the image, rather than the image itself in the database.

Setting Up the Database

Before you can begin using PHP with MySQL, you need a database. The `\Chapter 07\Data` folder of the source code located on the Sybex Web site contains a SQL script called `AmazonData.SQL`. Copy this file into the `\MySQL\bin` folder of your server (assuming you used a default setup). At the command line, type **`MySQL MySQL < AmazonData.SQL`** and press Enter. Your system will pause for a moment and return to the command prompt. That's all you need to do to create the database for this example.

You can use the MySQL utility to verify the presence of the database, table, and data at the command line. Simply type **`USE AmazonData;`** and press Enter. If the `AmazonData` database is present, the utility will use that database. Type **`SELECT * FROM DataStore`** and press Enter. You'll see a lot of data stream by if the script successfully created the table and filled it with data. An easier way to achieve the same results on a Windows system is to use the `WinMySQLadmin` utility shown in Figure 7.4. Simply select the Databases tab and you'll see the database, table, and associated fields.

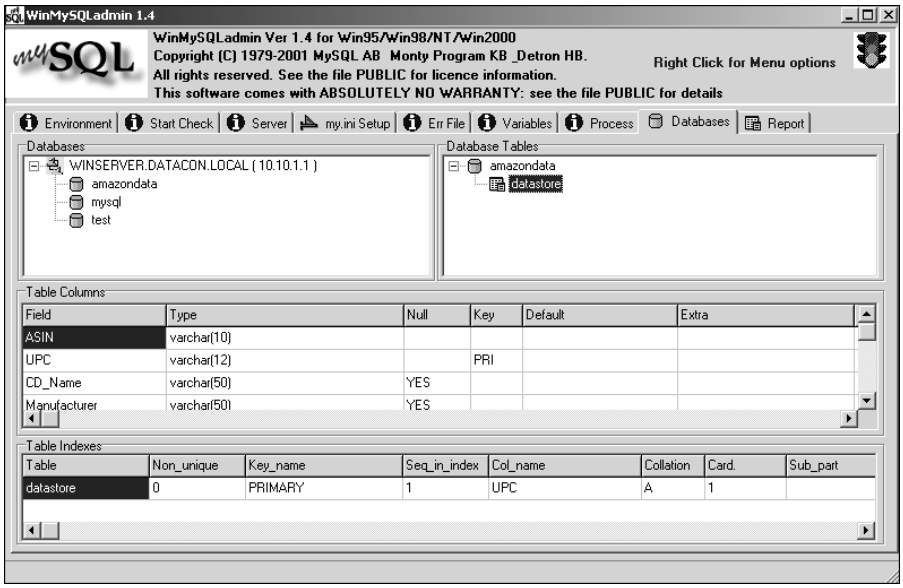
Writing the Sample Application

This example application requires five processing phases. First, you must create the local variables and determine whether the user has passed any information as part of the query. This part of the process works precisely the same as the processing described in Listing 7.1.

▶ TIP

Even though the example in this chapter stores the URL of the local copy of the image, you can replicate the functionality of the example in Chapter 6 almost exactly by adding a third party library to your PHP setup. This technique requires a platform-specific library and can cause browser compatibility issues in some cases. Because of these issues, I chose to discuss techniques that will work on most platforms with most browsers. The Dev Shed article at `http://www.devshed.com/Server_Side/PHP/ImageGeneration/page1.html` discusses the PHP image generation technique in detail.

FIGURE 7.4: The `WinMySQLadmin` utility validates the success of the script on a Windows machine.



The second phase is to display the HTML on screen. The request process works just the same as the code in Listing 7.2. The difference is that the request screen is the only thing that the user sees during the initial request. Only after the user clicks Submit does the return data appear. The reason for this change is that you don't know where to get the data from until the user makes the request.

The third phase detects the type of processing the request requires. In this case, the code determines whether the data appears in the local database or it needs to request the data from Amazon. Listing 7.6 shows this phase. You'll find this example in the `\Chapter 07\MySQLDemo` folder of the source code located on the Sybex Web site.

Listing 7.6 Determining How to Process a Request

```
// Create a database connection.
mysql_connect("localhost")
or die ("Cannot connect to the database.");

// Select the database.
mysql_select_db("AmazonData")
or die ("The database doesn't exist or is inaccessible.");

// Obtain the data.
$Output = mysql_query("SELECT * FROM DataStore WHERE UPC = '$UPC'");

// Verify the data is in the database.
```

```
if (mysql_num_rows ($Output) == 0)
   $Row = GetData();

else
{
   // Get the display data.
   $Row = mysql_fetch_row($Output)
   or die ("No Data in Query Row.");
}

// Get the current date and convert it to a timestamp.
$ConvDate = strtotime($Row[9]);

// Add 24 hours.
$ConvDate = $ConvDate + 86400;

// Verify the data isn't too old.
if ($ConvDate < time())
   $Row = GetData();
```

The code begins by using the `mysql_connect()` function to create a connection to the MySQL server. Depending on how you configure your database, you might also need to provide a username and password (or request this information from the user). The `mysql_select_db()` function to connect to the `AmazonData` database—this is the database that contains the information for the example. Finally, the code uses the `mysql_query()` function to obtain the data from the database if it exists.

The first check of the data determines whether the query returned any rows using the `mysql_num_rows()` function. If `$Output` is blank, then the UPC doesn't reside in the database and the code requests it using the `GetData()` function described later in this section. Even if `$Output` contains the requested data, there's no guarantee that this data is current. The next code check verifies the date the application last downloaded the data from Amazon. If the data is too old, then the code calls `GetData()` to retrieve the data.

The act of verification often leads into an optional fourth processing phase. The code might have to retrieve the data from Amazon. Listing 7.7 shows a straightforward SOAP technique you can use to perform this task without relying on the `AmazonSearch.PDF` file used for the example in the "Using the Amazon Supplied File Example" section.

Listing 7.7 Querying Amazon for Data Using SOAP

```
function GetData()
{
   // Access the global variables.
   global $UPC;
   global $Mode;
   global $Type;
```

```
global $DevTag;

// Create a new SOAP client.
$Client = new
soapclient("http://soap.amazon.com/schemas3/AmazonWebServices.wsdl",
true);

// This proxy helps you access the Amazon WSDL methods directly.
$Proxy = $Client->getProxy();

// Create an array to hold the request values.
$Parameters = array
(
   'upc'    => $UPC,
   'mode'   => $Mode,
   'tag'    => 'webservices-20',
   'type'   => $Type,
   'devtag' => $DevTag
);

// Invoke the Amazon method.
$AllData = $Proxy->UpcSearchRequest($Parameters);

// Retrieve the Details.
$DetailsNode = $AllData["Details"];
$Details = $DetailsNode[0];

// Get the image from online.
$ImageData = file($Details["ImageUrlMedium"]);

// Create a filename based on the existing filename.
$PathArray = parse_url($Details["ImageUrlMedium"]);
$Path = $PathArray["path"];
$FileArray = pathinfo($Path);
$File = $FileArray["basename"];

// Save the file on disk.
$FilePointer = fopen($File, "w");
for ($Counter = 0; $Counter < count($ImageData); $Counter++)
   fwrite($FilePointer, $ImageData[$Counter]);

// Remove the existing record (if any).
mysql_query('DELETE FROM DataStore WHERE UPC = "' . $UPC . '"');

// Get the scanning time.
$ScanTime = strftime("%y/%m/%d %H:%M:%S");

// Convert the release date.
$ReleaseTime = strftime("%y/%m/%d %H:%M:%S",
```

```
      strtotime(ereg_replace(",", " ", $Details["ReleaseDate"])));

   // Create the query string.
   $UpdateQuery =
      'INSERT INTO DataStore (ASIN, UPC, CD_Name, Manufacturer,
         Availability, Product_URL, List_Price, Release_Date, Picture,
         Scanned)
      VALUES ("' . $Details["Asin"] . '" , "' . $UPC . '" , "' .
         $Details["ProductName"] . '" , "' . $Details["Manufacturer"]
         . '" , "' . $Details["Availability"] . '" , "' .
         $Details["Url"] . '" , "' . substr($Details["ListPrice"], 1)
         . '" , "' . $ReleaseTime . '" , "' . $File . '" , "' .
         $ScanTime . '")';

   // Update the database.
   mysql_query($UpdateQuery)
   or die ("Error Updating Database");

   $Result = array($Details["Asin"], $UPC, $Details["ProductName"],
                   $Details["Manufacturer"], $Details["Availability"],
                   $Details["Url"], $Details["ListPrice"],
                   $Details["ReleaseDate"], $File, $ScanTime);
   return $Result;
}
```

The code begins by creating a SOAP client using the `soapclient()` constructor. This URL points to the same WSDL page that other example in this book uses. Once the code accesses the WSDL page, it creates a proxy for it so that the application can access Amazon Web Services. At this point, you can begin making requests of the service.

The code creates a specially formatted array to hold the request data. The capitalization of the array elements follows the capitalization in the Amazon Web Services Kit help file. It appears that these entries are case sensitive. Once the code creates a request array, it uses it to make the request using the `$Proxy->UpcSearchRequest()` method.

The request returns the same `ProductInfo` XML data as usual, so you need to parse through the various levels to find the `Details` node. Note that PHP treats the XML nodes as arrays, so you can use the same techniques that you use with any other array to access data members.

It's important to download the image for the product so the user can see it. To perform this task, the code opens the required image using the `file()` function. It then creates a local file with the same name with the `fopen()` function and uses the `fwrite()` function to store the image to that file. Make sure you use binary safe function calls to perform this task. The technique shown might not work with older versions of PHP. The documentation tells which versions support binary files.

At this point, the code has everything it needs to store the data in the table, so it uses the `mysql_query()` function to delete the existing record, if necessary. The call doesn't fail if the table doesn't contain the record, so you don't lose anything by making it. You'll find that MySQL is a little fussy about time formats and unfortunately, those formats aren't necessarily the same formats that PHP accepts. Consequently, the code performs some conversion of the two dates that the table must store—the released date and the date that the application last requested the data from Amazon. The final step is to insert the new record into the table using the `mysql_query()` function. Make sure you pay special attention to how the code creates `$UpdateQuery`. If you don't format the string correctly, the table stores incorrect data. For example, notice how the code removes the dollar sign from the incoming list price. The database stores a 0 for the list price if you don't perform this task.

The final step creates a `$Result` array from the data retrieved from Amazon and returns it to the caller for display. Notice that the code calls on the `$Details` array several times for the same data. In general, it's easier to use the technique shown to create the array, but you could possibly increase performance by storing the data in local single variables rather than read them from arrays every time you need them.

The fifth processing phase displays the data on screen. Like the other examples in the chapter, this phase uses the `print()` function to output a combination of HTML and data from the variables. The only surprises here are those that deal with differences in the way Amazon returns the data and the method used to store the data in the database. You need to detect the display issues and handle them in code as shown here.

```
// Display the list price.
print("<label>List Price: ");
if (substr($Row[6], 0, 1) <> "$")
   print("$");
print($Row[6]);
print("</label></br>");

// Display the release date.
print("<label>Release Date: ");
if (strstr($Row[7], "-"))
   print strftime("%d %B, %Y", strtotime($Row[7]));
else
   print($Row[7]);
print("</label></br>");
```

As you can see, the code detects the problem areas by looking for key strings within the variable. The database stores financial values without the dollar sign (or other monetary symbol), so the code adds it back in for a local response. Likewise, the database stores the date in a different format from Amazon, so the code needs to translate the database format to make it match the Amazon output. Figure 7.5 shows the output from this example.

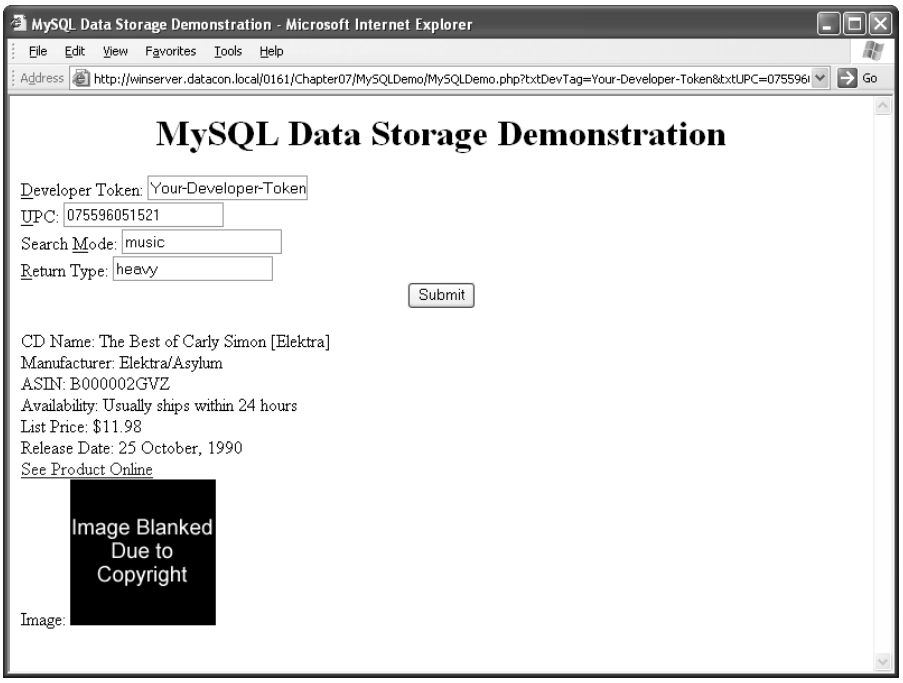


FIGURE 7.5: This application demonstrates the use of MySQL storage for Amazon data.

Your Call to Action

This chapter has helped you understand some of the essentials for using PHP with Amazon Web Services. PHP is a good solution for developers who want to create a capable Web site without relying on heavy amounts of scripting. In addition, PHP is the most portable solution you can use to build Amazon Web Services applications. Of course, this solution only works for Web applications—not for desktop applications such as those found in Chapter 6. In general, you can use other alternatives to get the same results with less coding or effort. Finally, PHP is a good solution for developers with a tight budget because you can get the required software free.

At this point, you have to decide whether you want to use a product like PHP to develop your next Web application. Most developers find PHP easy to work with and the price is right. Once you do decide to use PHP, make sure you get a good start by checking out the tutorials and other user help information in this chapter. Expert PHP developers can start writing their Amazon Web Services application immediately. The chapter shows several examples that demonstrate applications of varying complexity.

Chapter 8 moves from PHP to Java. You'll find that Java is another extremely popular choice and that the Amazon Web Services Kit provides good support for this language. Although Java isn't quite as easy as PHP to transfer from one platform to another, it's a more popular choice. In addition, you'll find that Java applications run faster because Java performs some of the interpretation required to run the application during a compile cycle. Java still doesn't run as fast as a native code application, but it's a very good choice.