Sams Teach Yourself PHP, MySQL and Apache All in One

Julie C. Meloni

SAMS
Introduction

Welcome to *Sams Teach Yourself PHP, MySQL and Apache All in One, Fourth Edition*! In the previous edition, the majority of the modifications were around the ability to use both PHP 5 and MySQL 5 as core technologies. In the two years since the previous edition was released, little has changed: PHP 5 and MySQL 5 are solid, stable, and power a great number of the Web-based applications we use every day. Many hosting providers now offer PHP 5 by default, without support for PHP 4 (which has entered the “end of life” phase), thus ensuring that anyone who wants to use PHP 5 and MySQL 5 can do so without installing these applications on their own (although the first four chapters of this book explain how to do just that). All the code in this edition is based on PHP 5 and, where appropriate, the MySQL Improved Extension (mysqli) in PHP, using MySQL 5 as the back-end database.

Some of you might have heard of PHP 6 or have seen books touting PHP 6 as the core language used. As of this writing in May of 2008, PHP 6 is still in the development stages and has not even entered the release candidate stage of development. Although PHP 6 is likely to reach the release candidate stage before the end of 2008, hosting providers will be loath to provide new technologies for general use until the language and the engine driving it have been thoroughly tested and improved to the point at which the release is deemed stable and mature—perhaps sometime in 2009. Given this information, it seemed entirely premature for this edition to cover the aspects of the language based on a developmental release, especially when the goal of this book is to provide the concepts necessary to master the basics of programming in the PHP language—the version that is stable and widely distributed.

Over the course of this book, you’ll learn the concepts necessary for configuring and managing the Apache web server, the basics of programming in PHP, and the methods for using and administering the MySQL relational database system. The overall goal of the book is to provide you with the foundation you need to understand how seamlessly these technologies integrate with one another and to give you practical knowledge of how to integrate them.

Who Should Read This Book?

This book is geared toward individuals who possess a general understanding of the concepts of working in a web-based development environment, be it Linux/UNIX, Windows, or even Mac OS X. Installation and configuration instructions assume that you have familiarity with your operating system and the basic methods of building (on Linux/UNIX systems) or installing (on Windows and Mac OS X systems) software.
The lessons that delve into programming with PHP assume no previous knowledge of the language. However, if you have experience with other programming languages, such as ASP (Active Server Pages), JSP (Java Server Pages), or Perl, you will find the going much easier. Similarly, if you have worked with other databases, such as Oracle or Microsoft SQL Server, you will already possess a solid foundation for working through the MySQL-related lessons.

The only real requirement is that you already understand static web content creation with HTML. If you are just starting out in the world of web development, you will still be able to use this book, but you should consider working through an HTML tutorial. If you are comfortable creating basic pages, you will be fine.

**How This Book Is Organized**

This book is divided into six parts, corresponding to particular topic groups. You should read the chapters within each part one right after another, with each chapter building on the information found in those before it:

- **Part I, “Getting Up and Running,”** provides a quick start guide to installation and walks you through the installation and configuration of MySQL, Apache, and PHP in depth. You’ll need to complete at least one version of these instructions—either the quick start installation or the longer instructions—before moving on unless you already have access to a working installation of these technologies through a hosting provider. Even if you don’t need to install and configure MySQL, Apache, and PHP in your development environment, you should still skim these lessons so that you understand the basics of their interaction.

- **Part II, “PHP Language Structure,”** is devoted to teaching you the basics of the PHP language, including structural elements such as arrays and objects. The examples will get you in the habit of writing code, uploading it to your server, and testing the results.

- **Part III, “Getting Involved with the Code,”** consists of chapters that cover intermediate-level application development topics, including working with forms and files, restricting access, and completing other small projects designed to introduce a specific concept.

- **Part IV, “PHP and MySQL Integration,”** contains chapters devoted to working with databases in general, such as database normalization, as well as using PHP to connect to and work with MySQL. Included is a basic SQL primer, which also includes MySQL-specific functions and other information.
Part V, “Basic Projects,” consists of chapters devoted to performing a particular task using PHP and MySQL, integrating all the knowledge gained so far. Projects include an address book, a discussion forum, and a basic online storefront, among others. These examples are built in a black-and-white environment, meaning the aesthetic display is minimal. This allows you to focus on the programming and logic involved in building the structures rather than making these items aesthetically pleasing.

Part VI, “Administration and Fine-Tuning,” is devoted to administering and tuning Apache and MySQL. It also includes information on virtual hosting and setting up a secure web server.

If you find that you are already familiar with a topic, you can skip ahead to the next chapter. However, in some instances, chapters refer to specific concepts learned in previous chapters, so be aware that you might have to skim a skipped chapter so that your development environment remains consistent with the book.

At the end of many chapters, a few quiz questions test how well you’ve learned the material. Additional activities provide another way to apply the information learned in the chapter and guide you toward using this newfound knowledge in the next chapter.

About the Book’s Source Code

All of the code that appears in listings throughout the chapters is also available on the accompanying CD-ROM. You may also download the source code bundle from the author’s website at http://www.thickbook.com/.

Typing the code on your own provides useful experience in making typos, causing errors, and performing the sometimes mind-numbing task of tracking down errant semicolons. However, if you want to skip that lesson and just upload the working code to your website, feel free!

Conventions Used in This Book

This book uses different typefaces to differentiate between code and plain English, and to help you identify important concepts. Throughout the chapters, code, commands, and text you type or see onscreen appear in a computer typeface. New terms appear in italics at the point in the text where they are defined. Additionally, icons accompany special blocks of information:
| **By the Way** | A “By the Way” note presents an interesting piece of information related to the current topic. |
| **Did you Know?** | A “Did You Know” tip offers advice or teaches an easier method for performing a task. |
| **Watch Out!** | A “Watch Out!” warns you about potential pitfalls and explains how to avoid them. |
The Building Blocks of PHP

In this chapter, you will get your hands dirty with some of the nuts and bolts of the PHP scripting language. Those of you new to programming might feel overwhelmed at times, but don’t worry—you can always refer to this chapter later on. Concentrate on understanding the concepts, rather than memorizing the features covered, because these elements will be repeated throughout the scripts in this book. Eventually you’ll get it, if not the first time!

If you’re already an experienced programmer, you should at least skim this chapter because it covers a few PHP-specific features with regards to global variables, data types, and changing types.

In this chapter, you will learn

- About variables—what they are, why you need to use them, and how to use them
- How to define and access variables
- About data types
- About some of the more commonly used operators
- How to use operators to create expressions
- How to define and use constants

Variables

A variable is a special container that you can define, which will then “hold” a value, such as a number, string, object, array, or a Boolean. Variables are fundamental to programming. Without variables, you would be forced to hard-code each specific value used in your scripts. The following hard-coded statement adds two numbers together and prints the result, which solves a simple mathematics problem:

```
echo (2 + 4);
```
However, this snippet of code is useful only for people who specifically want to know the sum of 2 and 4. To get past this limitation, you could write a script for finding the sum of another set of numbers, say 3 and 5. However, this approach to programming is clearly absurd, and this is where variables come into play.

Variables allow you to create templates for operations, such as adding two numbers, without worrying about the specific values the variables represent. Values will be given to the variables when the script is run, possibly through user input, through a database query, or from the result of another action earlier in the script. In other words, variables should be used whenever the data in your script is liable to change—either during the lifetime of the script, or when it is passed to another script for later use.

A variable consists of a name of your choosing, preceded by a dollar sign ($). Variable names can include letters, numbers, and the underscore character (_), but they cannot include spaces. Names must begin with a letter or an underscore. The following list shows some legal variables:

```
$a;
$a_longish_variable_name;
$2453;
$sleepyZZZZ;
```

Your variable names should be meaningful as well as consistent in style. For example, if your script deals with name and password values, don’t create a variable called $n for the name and $p for the password—those are not meaningful names for anyone other than you, at that particular moment. If you pick up that script weeks later, you might think that $n is the variable for “number” rather than “name” and that $p stands for “page” rather than “password.” And what if a co-worker has to modify your script? How will that person know what $n and $p stood for? You can use whatever naming convention you want for variables in your scripts, as long as the names are descriptive and follow some sort of pattern that others can understand.

A semicolon (;)—also known as the instruction terminator—is used to end a PHP statement. The semicolons in the previous fragment of code are not part of the variable names, but are used to end the statement that declares the variable as “alive and kicking,” if you will. To declare a variable, you need only include it in your
script. When you declare a variable, you usually assign a value to it in the same statement, as shown here:

```php
$num1 = 8;
$num2 = 23;
```

The preceding lines declare two variables and use the assignment operator (=) to assign values to them. You will learn about assignment in more detail in the “Operators and Expressions” section later in this chapter. After you assign values to your variables, you can treat them exactly as if they were the values themselves. In other words

```php
echo $num1;
```

is equivalent to

```php
echo 8;
```

as long as `$num1` is assigned a value of 8.

### Globals and Superglobals

In addition to the rules for naming variables, there are rules regarding the availability of variables. In general, the assigned value of a variable is present only within the function or script where it resides. For example, if you have `scriptA.php` that holds a variable called `$name` with a value of `joe`, and you want to create `scriptB.php` that also uses a `$name` variable, you can assign to that second `$name` variable a value of `jane` without affecting the variable in `scriptA.php`. The value of the `$name` variable is local to each script, and the assigned values are independent of each other.

However, you can also define the `$name` variable as global within a script or function. If the `$name` variable is defined as a global variable in both `scriptA.php` and `scriptB.php`, and these scripts are connected to each other (that is, one script calls the other or includes the other), there will only be one value for the now-shared `$name` variable. Examples of global variable scope will be explained in more detail in Chapter 7, “Working with Functions.”

In addition to global variables of your own creation, PHP has several predefined variables called superglobals. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

- `$_GET` contains any variables provided to a script through the GET method.
- `$_POST` contains any variables provided to a script through the POST method.
The Building Blocks of PHP

- $_COOKIE contains any variables provided to a script through a cookie.
- $_FILES contains any variables provided to a script through file uploads.
- $_SERVER contains information such as headers, file paths, and script locations.
- $_ENV contains any variables provided to a script as part of the server environment.
- $_REQUEST contains any variables provided to a script via GET, POST, or COOKIE input mechanisms.
- $_SESSION contains any variables that are currently registered in a session.

The examples in this book will use superglobals wherever possible. Using superglobals within your scripts is important in creating secure applications because superglobals reduce the likelihood of user-injected input to your scripts. By coding your scripts to accept only what you want, in the manner defined by you (from a form using the POST method, or from a session, for example), you can eliminate some of the problems created by loosely written scripts.

Data Types

Different types of data take up different amounts of memory and may be treated differently when they are manipulated in a script. Some programming languages therefore demand that the programmer declare in advance which type of data a variable will contain. By contrast, PHP is loosely typed, meaning that it will determine the data type at the time data is assigned to each variable.

This automatic typing is a mixed blessing. On the one hand, it means that variables can be used flexibly—in one instance, a variable can hold a string and then later in the script it can hold an integer or some other data type. On the other hand, this flexibility can lead to problems in larger scripts if you are specifically expecting a variable to hold one data type when in fact it holds something completely different. For example, suppose that you have created code to manipulate an array variable. If the variable in question instead contains a number value and no array structure is in place, errors will occur when the code attempts to perform array-specific operations on the variable.

Table 5.1 shows the eight standard data types available in PHP.
**TABLE 5.1 Standard Data Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>One of the special values true or false</td>
</tr>
<tr>
<td>Integer</td>
<td>5</td>
<td>A whole number</td>
</tr>
<tr>
<td>Float or double</td>
<td>3.234</td>
<td>A floating-point number</td>
</tr>
<tr>
<td>String</td>
<td>&quot;hello&quot;</td>
<td>A collection of characters</td>
</tr>
<tr>
<td>Object</td>
<td></td>
<td>An instance of a class</td>
</tr>
<tr>
<td>Array</td>
<td></td>
<td>An ordered set of keys and values</td>
</tr>
<tr>
<td>Resource</td>
<td></td>
<td>Reference to a third-party resource (a database, for example)</td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td>An uninitialized variable</td>
</tr>
</tbody>
</table>

Resource types are often returned by functions that deal with external applications or files. For example, you will see references to “the MySQL resource ID” in Chapter 18, “Interacting with MySQL Using PHP.” The NULL type is reserved for variables that have been declared, but no value has been assigned to them.

PHP has several functions available to test the validity of a particular type of variable—one for each type, in fact. The is_* family of functions, such as is_bool(), tests whether a given value is a Boolean. Listing 5.1 assigns different data types to a single variable and then tests the variable with the appropriate is_* function. The comments in the code show you where the script is in the process.

You can read more about calling functions in Chapter 7.

---

**LISTING 5.1 Testing the Type of a Variable**

```php
1: <?php
2:  $testing; // declare without assigning
3:  echo "is null? ".is_null($testing); // checks if null
4:  echo "<br/>";
5:  $testing = 5;
6:  echo "is an integer? ".is_int($testing); // checks if integer
7:  echo "<br/>";
8:  $testing = "five";
9:  echo "is a string? ".is_string($testing); // checks if string
10: echo "<br/>";
11: $testing = 5.024;
12: echo "is a double? ".is_double($testing); // checks if double
13: echo "<br/>";
14: $testing = true;
15: echo "is boolean? ".is_bool($testing); // checks if boolean
16: echo "<br/>";
17: $testing = array('apple', 'orange', 'pear');
```
LISTING 5.1  Continued

18: echo "is an array? ".is_array($testing); // checks if array
19: echo "<br/>";
20: echo "is numeric? ".is_numeric($testing); // checks if is numeric
21: echo "<br/>";
22: echo "is a resource? ".is_resource($testing); // checks if is a resource
23: echo "<br/>";
24: echo "is an array? ".is_array($testing); // checks if is an array
25: echo "<br/>";
26: ?>

Put these lines into a text file called testtype.php, and place this file in your web server document root. When you access this script through your web browser, it produces the following output:

is null? 1
is an integer? 1
is a string? 1
is a double? 1
is boolean? 1
is an array? 1
is numeric?
is a resource?
is an array? 1

When the $testing variable is declared in line 2, no value is assigned to it, so when the variable is tested in line 3 to see whether it is null (using is_null()), the result is 1 (true). After this, values are assigned to $testing by using the = sign before testing it with the appropriate is_* function. An integer, assigned to the $testing variable in line 5, is a whole or real number. In simple terms, you can think of a whole number as a number without a decimal point. A string, assigned to the $testing variable in line 8, is a collection of characters. When you work with strings in your scripts, they should always be surrounded by double or single quotation marks (" or "). A double, assigned to the $testing variable in line 11, is a floating-point number (that is, a number that includes a decimal point). A Boolean, assigned to the $testing variable in line 14, can have one of two special values: true or false. In line 17, an array is created using the array() function, which you’ll learn more about in Chapter 8, “Working with Arrays.” This particular array contains three items, and the script dutifully reports $testing to have a type of “array.”

From line 20 through the end of the script, no value is reassigned to $testing—only the type is tested. Lines 20 and 22 test whether $testing is a numeric or resource type, respectively, and because it is not, no value is displayed to the user. In line 24, the script tests again to see whether $testing is an array, and because it is, the value of 1 is displayed.
Changing Type with `settype()`

PHP also provides the function `settype()`, which is used to change the type of a variable. To use `settype()`, you place the variable to change and the type to change it to between the parentheses and separate the elements with a comma, like this:

```
settype($variabletochange, 'new type');
```

Listing 5.2 converts the value 3.14 (a float) to each of the four standard types examined in this chapter.

Listing 5.2 Changing the Type of a Variable with `settype()`

```
1:  <?php
2:  $undecided = 3.14;
3:  echo "is ".$undecided." a double? ".is_double($undecided)."<br/>"; // double
4:  settype($undecided, 'string');
5:  echo "is ".$undecided." a string? ".is_string($undecided)."<br/>"; // string
6:  settype($undecided, 'integer');
7:  echo "is ".$undecided." an integer? ".is_integer($undecided)."<br/>"; // int
8:  settype($undecided, 'double');
9:  echo "is ".$undecided." a double? ".is_double($undecided)."<br/>"; // double
10: settype($undecided, 'bool');
11: echo "is ".$undecided." a boolean? ".is_bool($undecided)."<br/>"; // boolean
12: ?>
```

Per the PHP Manual, “double” is returned in case of a float, and not simply “float”. Your eyes are not deceiving you.

In each case, we use the appropriate `is_*` function to confirm the new data type and to print the value of the variable `$undecided` to the browser using `echo`. When we convert the string “3.14” to an integer in line 6, any information beyond the decimal point is lost forever. That’s why `$undecided` contains 3 after we change it back to a double in line 8. Finally, in line 10, we convert `$undecided` to a Boolean. Any number other than 0 becomes `true` when converted to a Boolean. When printing a Boolean in PHP, `true` is represented as 1 and `false` is represented as an empty string, so in line 11, `$undecided` is printed as 1.

Put these lines into a text file called `settype.php` and place this file in your web server document root. When you access this script through your web browser, it produces the following output:

```
is 3.14 a double? 1
is 3.14 a string? 1
is 3 an integer? 1
is 3 a double? 1
is 1 a boolean? 1
```


**Changing Type by Casting**

The principal difference between using `settype()` to change the type of an existing variable and changing type by *casting* is the fact that casting produces a copy, leaving the original variable untouched. To change type through casting, you indicate the name of a data type, in parentheses, in front of the variable you are copying. For example, the following line creates a copy of the `$originalvar` variable, with a specific type (integer) and a new name `$newvar`. The `$originalvar` variable will still be available, and will be its original type; `$newvar` is a completely new variable.

```
$newvar = (integer) $originalvar
```

Listing 5.3 illustrates changing data types through casting.

**LISTING 5.3  Casting a Variable**

```php
<?php
$undecided = 3.14;
$holder = (double) $undecided;
echo "is 
$holder
" a double? 
.is_double($holder)."
<br/>"; // double
$holder = (string) $undecided;
echo "is 
$holder
" a string? 
.is_string($holder)."
<br/>"; // string
$holder = (integer) $undecided;
echo "is 
$holder
" an integer? 
.is_integer($holder)."
<br/>"; // integer
$holder = (double) $undecided;
echo "is 
$holder
" a double? 
.is_double($holder)."
<br/>"; // double
$holder = (boolean) $undecided;
echo "is 
$holder
" a boolean? 
.is_bool($holder)."
<br/>"; // boolean
echo "<br/>
";
echo "original variable type of $undecided: 
";
echo gettype($undecided); // double
?>
```

Listing 5.3 never actually changes the type of the `$undecided` variable, which remains a double throughout this script, as illustrated on line 15, where the `gettype()` function is used to determine the type of `$undecided`.

Despite its usage here, don’t use `gettype()` to test for a certain type because it can be slow and is likely to be deprecated in future versions. Use the `is_*` family of functions to test type in production. This usage is simply for illustrative purposes.

In fact, casting `$undecided` creates a copy that is then converted to the type specified at the time of the cast, and stored in the variable `$holder`. This casting occurs first in line 3, and again in lines 5, 7, 9, and 11. Because the code is working with only a copy of `$undecided` and not the original variable, it never lost its original value, as the `$undecided` variable did in line 6 of Listing 5.2 when its type changed from a string to an integer.
Put the contents of Listing 5.3 into a text file called casttype.php and place this file in your web server document root. When you access this script through your web browser, it produces the following output:

is 3.14 a double? 1
is 3.14 a string? 1
is 3 an integer? 1
is 3.14 a double? 1
is 1 a boolean? 1
original variable type of 3.14: double

Now that you’ve seen how to change the contents of a variable from one type to another either by using `settype()` or by casting, consider why this might be useful. It is not a procedure that you will have to use often because PHP automatically casts your variables for you when the context of the script requires a change. However, such an automatic cast is temporary, and you might want to make a variable persistently hold a particular data type—thus, the ability to specifically change types.

For example, the numbers that a user types into an HTML form will be made available to your script as the string type. If you try to add two strings together because they contain numbers, PHP will helpfully convert these strings into numbers while the addition is taking place. So

```
"30cm" + "40cm"
```

results in an answer of 70.

The generic term *number* is used here to mean integers and floats. If the user input is in float form, and the strings added together were "3.14cm" and "4.12cm", the answer provided would be 7.26.

During the casting of a string into an integer or float, PHP will ignore any non-numeric characters. The string will be truncated, and any characters from the location of the first non-numeric character onward are ignored. So, whereas "30cm" is transformed into "30", the string "6ft2in" becomes just 6 because the rest of the string evaluates to zero.

You might want to clean up the user input yourself and use it in a particular way in your script. Imagine that the user has been asked to submit a number. We can simulate this by declaring a variable and assigning the user’s input to it:

```php
$test = "30cm";
```
As you can see, the user has added units to his number—instead of entering "30", the user has entered "30cm". You can make sure that the user input is clean by casting it as an integer:

```php
$newtest = (integer) $test;
echo "Your imaginary box has a width of $newtest centimeters.";
```

The resulting output would be

Your imaginary box has a width of 30 centimeters.

Had the the user input not been cast, and the value of the original variable, $test, been used in place of $newtest when printing the statement regarding the width of a box, the result would have been

Your imaginary box has a width of 30cm centimeters.

This output looks strange; in fact, it looks like parroted user input that hadn’t been cleaned up (which is exactly what it is).

**Why Test Type?**

Why might it be useful to know the type of a variable? There are often circumstances in programming in which data is passed to you from another source. In Chapter 7, you will learn how to create functions in your scripts, and data is often passed between one or more functions because they can accept information from calling code in the form of arguments. For the function to work with the data it is given, it is a good idea to first verify that the function has been given values of the correct data type. For example, a function expecting data that has a type of “resource” will not work well when passed a string.

**Operators and Expressions**

With what you have learned so far, you can assign data to variables, and you can even investigate and change the data type of a variable. A programming language isn’t very useful, though, unless you can manipulate the data you have stored. **Operators** are symbols used to manipulate data stored in variables, to make it possible to use one or more values to produce a new value, or to check the validity of data to determine the next step in a condition, and so forth. A value operated on by an operator is referred to as an **operand**.
An operator is a symbol or series of symbols that, when used in conjunction with values, performs an action and usually produces a new value. An operand is a value used in conjunction with an operator. There are usually two or more operands to one operator.

In this simple example, two operands are combined with an operator to produce a new value:

\[(4 + 5)\]

The integers 4 and 5 are operands. The addition operator (+) operates on these operands to produce the integer 9. Operators almost always sit between two operands, although you will see a few exceptions later in this chapter.

The combination of operands with an operator to produce a result is called an expression. Although operators and their operands form the basis of expressions, an expression need not contain an operator. In fact, an expression in PHP is defined as anything that can be used as a value. This includes integer constants such as 654, variables such as $user, and function calls such as gettype(). The expression \((4 + 5)\), for example, consists of two expressions (4 and 5) and an operator (+). When an expression produces a value, it is often said to resolve to that value. That is, when all subexpressions are taken into account, the expression can be treated as if it were a code for the value itself. In this case, the expression \((4 + 5)\) resolves to 9.

An expression is any combination of functions, values, and operators that resolves to a value. As a rule of thumb, if you can use it as if it were a value, it is an expression.

Now that you have the principles out of the way, it's time to take a tour of the operators commonly used in PHP programming.

**The Assignment Operator**

You have seen the assignment operator in use each time a variable was declared in an example; the assignment operator consists of the single character: =. The assignment operator takes the value of the right-side operand and assigns it to the left-side operand:

\[\$name = "jimbo";\]

The variable $name now contains the string "jimbo". This construct is also an expression. Although it might seem at first glance that the assignment operator
simply changes the variable $name without producing a value, in fact, a statement that uses the assignment operator always resolves to a copy of the value of the right operand. Thus

```php
echo $name = "jimbo";
```

prints the string "jimbo" to the browser while it also assigns the value "jimbo" to the $name variable.

### Arithmetic Operators

The arithmetic operators do exactly what you would expect—they perform arithmetic operations. Table 5.2 lists these operators along with examples of their usage and results.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Sample Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>10+3</td>
<td>13</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>10-3</td>
<td>7</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>10/3</td>
<td>3.3333333333333</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>10*3</td>
<td>30</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>10%3</td>
<td>1</td>
</tr>
</tbody>
</table>

The addition operator adds the right-side operand to the left-side operand. The subtraction operator subtracts the right-side operand from the left-side operand. The division operator divides the left-side operand by the right-side operand. The multiplication operator multiplies the left-side operand by the right-side operand. The modulus operator returns the remainder of the left-side operand divided by the right-side operand.

### The Concatenation Operator

The concatenation operator is represented by a single period (.). Treating both operands as strings, this operator appends the right-side operand to the left-side operand. So

```php
"hello"." world"
```

returns

"hello world"
Note that the resulting space between the words occurs because there is a leading space in the second operand (" world" instead of "world"). The concatenation operator literally smashes together two strings without adding any padding. So, if you tried to concatenate two strings without leading or trailing spaces, such as "hello"."world"
you would get this as your result:
"helloworld"

Regardless of the data types of the operands used with the concatenation operator, they are treated as strings, and the result will always be of the string type. You will encounter concatenation frequently throughout this book when the results of an expression of some kind must be combined with a string, as in

```
$cm = 212;
echo "the width is ".($cm/100)." meters";
```

### Combined Assignment Operators

Although there is only one true assignment operator, PHP provides a number of combination operators that transform the left-side operand and return a result, while also modifying the original value of the variable. As a rule, operators use operands but do not change their original values, but combined assignment operators break this rule. A combined assignment operator consists of a standard operator symbol followed by an equal sign. Combination assignment operators save you the trouble of using two operators in two different steps within your script. For example, if you have a variable with a value of 4, and you want to increase this value to 4 more, you might see:

```
$x = 4;
$x = $x + 4; // $x now equals 8
```

However, you can also use a combination assignment operator (+=) to add and return the new value, as shown here:

```
$x = 4;
$x += 4; // $x now equals 8
```

Each arithmetic operator, as well as the concatenation operator, also has a corresponding combination assignment operator. Table 5.3 lists these new operators and shows an example of their usage.
TABLE 5.3 Some Combined Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>$x += 5</td>
<td>$x = $x + 5</td>
</tr>
<tr>
<td>-=</td>
<td>$x -= 5</td>
<td>$x = $x - 5</td>
</tr>
<tr>
<td>/=</td>
<td>$x /= 5</td>
<td>$x = $x / 5</td>
</tr>
<tr>
<td>*=</td>
<td>$x *= 5</td>
<td>$x = $x * 5</td>
</tr>
<tr>
<td>%=</td>
<td>$x %= 5</td>
<td>$x = $x % 5</td>
</tr>
<tr>
<td>.=</td>
<td>$x .= &quot;test&quot;</td>
<td>$x = $x.&quot;test&quot;</td>
</tr>
</tbody>
</table>

Each of the examples in Table 5.3 transforms the value of $x using the value of the right-side operand. Subsequent uses of $x will refer to the new value. For example

```
$x = 4;
$x += 4; // $x now equals 8
$x += 4; // $x now equals 12
$x -= 3; // $x now equals 9
```

These operators will be used throughout the scripts in the book. You will frequently see the combined concatenation assignment operator when you begin to create dynamic text; looping through a script and adding content to a string, such as dynamically building the HTML code to represent a table, is a prime example of the use of a combined assignment operator.

**Automatically Incrementing and Decrementing an Integer Variable**

When coding in PHP, you will often find it necessary to increment or decrement a variable that is an integer type. You will usually need to do this when you are counting the iterations of a loop. You have already learned two ways of doing this—either by incrementing the value of $x using the addition operator

```
$x = $x + 1; // $x is incremented by 1
```

or by using a combined assignment operator

```
$x += 1; // $x is incremented by 1
```

In both cases, the new value is assigned to $x. Because expressions of this kind are common, PHP provides some special operators that allow you to add or subtract the integer constant 1 from an integer variable, assigning the result to the variable itself. These are known as the *post-increment* and *post-decrement* operators. The post-increment operator consists of two plus symbols appended to a variable name:

```
$x++; // $x is incremented by 1
```
This expression increments the value represented by the variable $x$ by one. Using two minus symbols in the same way will decrement the variable:

$x--; // x is decremented by 1

If you use the post-increment or post-decrement operators in conjunction with a conditional operator, the operand will be modified only after the first operation has finished:

$x = 3;
$y = $x++ + 3;

In this instance, $y$ first becomes 6 (the result of 3 + 3) and then $x$ is incremented.

In some circumstances, you might want to increment or decrement a variable in a test expression before the test is carried out. PHP provides the pre-increment and pre-decrement operators for this purpose. These operators behave in the same way as the post-increment and post-decrement operators, but they are written with the plus or minus symbols preceding the variable:

++$x; // $x is incremented by 1
--$x; // $x is decremented by 1

If these operators are used as part of a test expression, incrementing occurs before the test is carried out. For example, in the next fragment, $x$ is incremented before it is tested against 4.

$x = 3;
++$x < 4; // false

The test expression returns false because 4 is not smaller than 4.

**Comparison Operators**

Comparison operators perform comparative tests using their operands and return the Boolean value `true` if the test is successful or `false` if the test fails. This type of expression is useful when using control structures in your scripts, such as `if` and `while` statements. This book covers `if` and `while` statements in Chapter 6, “Flow Control Functions in PHP.”

For example, to test whether the value contained in $x$ is smaller than 5, you can use the less-than operator as part of your expression:

$x < 5
If $x$ contains the value 3, this expression will have the value true. If $x$ contains 7, the expression resolves to false.

Table 5.4 lists the comparison operators.

**TABLE 5.4** Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Returns True If...</th>
<th>Example ($x$ Is 4)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equivalence</td>
<td>Left is equivalent to right</td>
<td>$x == 5$</td>
<td>false</td>
</tr>
<tr>
<td>!=</td>
<td>Non-equivalence</td>
<td>Left is not equivalent to right</td>
<td>$x != 5$</td>
<td>true</td>
</tr>
<tr>
<td>===</td>
<td>Identical</td>
<td>Left is equivalent to right and they are the same type</td>
<td>$x === 4$</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>Non-equivalence</td>
<td>Left is equivalent to right but they are not the same type</td>
<td>$x === &quot;4&quot;$</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>Left is greater than right</td>
<td>$x &gt; 4$</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>Left is greater than or equal to right</td>
<td>$x &gt;= 4$</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>Left is less than right</td>
<td>$x &lt; 4$</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>Left is less than or equal to right</td>
<td>$x &lt;= 4$</td>
<td>true</td>
</tr>
</tbody>
</table>

These operators are most commonly used with integers or doubles, although the equivalence operator is also used to compare strings. Be very sure to understand the difference between the == and = operators. The == operator tests equivalence, whereas the = operator assigns value. Also, remember that === tests equivalence with regards to both value and type.

**Creating Complex Test Expressions with the Logical Operators**

Logical operators test combinations of Boolean values. For example, the or operator, which is indicated by two pipe characters (| |) or simply the word or, returns the Boolean value true if either the left or the right operand is true:

true || false

This expression returns true.
The `and` operator, which is indicated by two ampersand characters (&&) or simply the word `and`, returns the Boolean value `true` only if both the left and right operands are true:

```
true && false
```

This expression returns the Boolean value `false`. It’s unlikely that you will use a logical operator to test Boolean constants because it makes more sense to test two or more expressions that resolve to a Boolean. For example:

```
($x > 2) && ($x < 15)
```

returns the Boolean value `true` if `$x` contains a value that is greater than 2 and smaller than 15. Parentheses are used when comparing expressions to make the code easier to read and to indicate the precedence of expression evaluation. Table 5.5 lists the logical operators.

### Table 5.5 Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Returns True If…</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Or</td>
<td>Left or right is true</td>
</tr>
<tr>
<td>or</td>
<td>Or</td>
<td>Left or right is true</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>xor</td>
<td>Xor</td>
<td>Left or right is true but not both</td>
<td>true xor true</td>
<td>false</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
<td>Left and right are true</td>
<td>true &amp;&amp; false</td>
<td>false</td>
</tr>
<tr>
<td>and</td>
<td>And</td>
<td>Left and right are true</td>
<td>true and false</td>
<td>false</td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
<td>The single operand is not true</td>
<td>! true</td>
<td>false</td>
</tr>
</tbody>
</table>

You might wonder why are there two versions of both the `or` and the `and` operators, and that’s a good question. The answer lies in operator precedence, which you will examine next.

### Operator Precedence

When you use an operator within an expression, the PHP engine usually reads your expression from left to right. For complex expressions that use more than one operator, though, the PHP engine could be led astray without some guidance. First, consider a simple case:

```
4 + 5
```
There’s no room for confusion here—PHP simply adds 4 to 5. But what about the following fragment, with two operators:

\[ 4 + 5 \times 2 \]

This presents a problem. Should PHP find the sum of 4 and 5, and then multiply it by 2, providing the result 18? Or does it mean 4 plus the result of 5 multiplied by 2, resolving to 14? If you were simply to read from left to right, the former would be true. However, PHP attaches different precedence to different operators, and because the multiplication operator has higher precedence than the addition operator, the second solution to the problem is the correct one: 4 plus the result of 5 multiplied by 2.

However, you can override operator precedence by putting parentheses around your expressions. In the following fragment, the addition expression will be evaluated before the multiplication expression:

\[ (4 + 5) \times 2 \]

Whatever the precedence of the operators in a complex expression, it is a good idea to use parentheses to make your code clearer and to save you from bugs such as applying sales tax to the wrong subtotal in a shopping cart situation. The following is a list of the operators covered in this chapter in precedence order (those with highest precedence are listed first):

\[ \text{++, --, (cast)} \]
\[ /, *, \% \]
\[ +, - \]
\[ <, <=, =>, > \]
\[ ==, ===, != \]
\[ && \]
\[ || \]
\[ =, +=, -=, /=, *=, %=, .= \]

and
xor
or

As you can see, or has a lower precedence than ||, and and has a lower precedence than &&, so you can use the lower-precedence logical operators to change the way a complex test expression is read. In the following fragment, the two expressions are equivalent, but the second is much easier to read:
$x$ and $y$ || $z$

$x$ && ($y$ || $z$)

Taking it one step further, the following fragment is easier still:

$x$ and ($y$ or $z$)

However, all three examples are equivalent.

The order of precedence is the only reason that both && and and are available in PHP. The same is true of || and or. In most circumstances, the use of parentheses makes for clearer code and fewer bugs than code that takes advantage of the difference in precedence of these operators. This book will tend to use the more common || and && operators, and rely on parenthetical statements to set specific operator precedence.

### Constants

Variables offer a flexible way of storing data because you can change their values and the type of data they store at any time during the execution of your scripts. However, if you want to work with a value that must remain unchanged throughout your script's execution, you can define and use a constant. You must use PHP's built-in `define()` function to create a constant, which subsequently cannot be changed unless you specifically `define()` it again. To use the `define()` function, place the name of the constant and the value you want to give it, within parentheses and separated by a comma:

```php
define("YOUR_CONSTANT_NAME", 42);
```

The value you want to set can be a number, a string, or a Boolean. By convention, the name of the constant should be in capital letters. Constants are accessed with the constant name only; no dollar symbol is required. Listing 5.4 shows you how to define and access a constant.

### Listing 5.4 Defining and Accessing a Constant

```php
<?php
define("THE_YEAR", "2008");
echo "It is the year ".THE_YEAR;
?>
```
Constants can be used anywhere in your scripts, including in functions stored in external files.

Notice that in line 3 the concatenation operator is used to append the value held by the constant to the string "It is the year " because PHP does not distinguish between a constant and a string within quotation marks.

Put these few lines into a text file called constant.php and place this file in your web server document root. When you access this script through your web browser, it produces the following output:

It is the year 2008

The define() function can also accept a third Boolean argument that determines whether the constant name should be case sensitive. By default, constant names are case sensitive. However, by passing true to the define() function, you can change this behavior, so if you were to set up our THE_YEAR constant as

```
define("THE_YEAR", "2008", true);
```

you could access its value without worrying about case:

```
echo the_year;
echo ThE_YeAr;
echo THE_YEAR;
```

The preceding three expressions are equivalent, and all would result in an output of 2008. This feature can make scripts a little friendlier for other programmers who work with our code because they will not need to consider case when accessing a constant we have already defined. On the other hand, given the fact that other constants are case sensitive, this might make for more, rather than less, confusion as programmers forget which constants to treat in which way. Unless you have a compelling reason to do otherwise, the safest course is to keep your constants case sensitive and define them using uppercase characters, which is an easy-to-remember (not to mention standard) convention.

**Predefined Constants**

PHP automatically provides some built-in constants for you. For example, the constant __FILE__ returns the name of the file that the PHP engine is currently reading. The constant __LINE__ returns the current line number of the file. These constants are useful for generating error messages. You can also find out which version of PHP is interpreting the script with the PHP_VERSION constant. This constant can be useful
if you need version information included in script output when sending a bug report.

**Summary**

This chapter covered some of the basic features of the PHP language. You learned about variables and how to assign values to them using the assignment operator, as well as received an introduction to the scope of variables and built-in superglobals. You got an introduction to operators and learned how to combine some of the most common of these into expressions. Finally, you learned how to define and access constants.

Now that you have mastered some of the fundamentals of PHP, the next chapter will really put you in the driver's seat. You will learn how to make scripts that can make decisions and repeat tasks, with help from variables, expressions, and operators.

**Q&A**

**Q.** *Why is it useful to know the type of data that a variable holds?*

**A.** Often the data type of a variable constrains what you can do with it. For example, you can’t perform array-related functions on simple strings. Similarly, you might want to make sure that a variable contains an integer or a float before using it in a mathematical calculation, even though PHP will often help you by changing data types for you in this situation.

**Q.** *Should I obey any conventions when naming variables?*

**A.** Your goal should always be to make your code easy to read and understand. A variable such as $ab123245 tells you nothing about its role in your script and invites typos. Keep your variable names short and descriptive.

A variable named $f is unlikely to mean much to you when you return to your code after a month or so. A variable named $filename, on the other hand, should make more sense.

**Q.** *Should I learn the operator precedence table?*

**A.** There is no reason you shouldn’t, but I would save the effort for more useful tasks. By using parentheses in your expressions, you can make your code easy to read while defining your own order of precedence.
Workshop

The workshop is designed to help you anticipate possible questions, review what you’ve learned, and begin putting your knowledge into practice.

Quiz

1. Which of the following variable names are not valid?
   - $a\_value\_submitted\_by\_a\_user
   - $666666xyz
   - $xyz066666
   - $_____counter_____
   - $the\_first
   - $file\_name

2. What will the following code fragment output?
   ```php
   $num = 33;
   (boolean) $num;
   echo $num;
   ```

3. What will the following statement output?
   ```php
   echo gettype("4");
   ```

4. What will be the output from the following code fragment?
   ```php
   $test\_val = 5.5466;
   settype($test\_val, "integer");
   echo $test\_val;
   ```

5. Which of the following statements does not contain an expression?
   - 4;
   - `gettype(44);
   - 5/12;

6. Which of the statements in question 5 contains an operator?

7. What value will the following expression return?
   - 5 < 2

   What data type will the returned value be?
**Answers**

1. The variable name `$666666xyz` is not valid because it does not begin with a letter or an underscore character. The variable name `$the_first` is not valid because it contains a space. `$file-name` is also invalid because it contains a nonalphanumeric character (-).

2. The fragment will print the integer 33. The cast to Boolean produces a converted copy of the value stored in `$num`. It does not alter the value actually stored there.

3. The statement will output the string "string".

4. The code will output the value 5. When a float is converted to an integer, any information beyond the decimal point is lost.

5. They are all expressions because they all resolve to values.

6. The statement `5/12;` contains a division operator.

7. The expression will resolve to `false`, which is a Boolean value.

**Activities**

1. Create a script that contains at least five different variables. Populate them with values of different data types and use the `gettype()` function to print each type to the browser.

2. Assign values to two variables. Use comparison operators to test whether the first value is
   - The same as the second
   - Less than the second
   - Greater than the second
   - Less than or equal to the second

Print the result of each test to the browser.

Change the values assigned to your test variables and run the script again.
SYMBOLS

&& (and) operators, 103
* (asterisks)
  multiplication operators, 98
  table creation, 313
  wildcards, 38
\ (backslashes)
  converting time stamps to dates, 195
  directives, 51
\n (newline) character, 124, 214, 250
$ COOKIE superglobal, 90, 225
$ FILES superglobal, 90, 217
$ GET superglobal, 89
$ POST superglobal, 89
$ POST value, 379
$ REQUEST superglobal, 90
$ SESSION superglobal, 90
$SESSION superglobal, 229-232, 235
$start variable, 464
$txt variable, 135
= (equal sign)
  assignment operators, 89, 97
  concatenation operators, 214
== (equivalence) operators, 102, 316, 322
=== (identical) operators, 102
! (not operators), 103
!= (nonequivalence) operators, 102, 316
> (greater than) operators, 102, 316

$display block strings
  online address books, 399, 402
  topic posts (discussion forums), 427
$display block value, 442-443
$display value, 439
$ENV superglobal, 90
$file array variable, 220
$file dir variable, 219
$file name variable, 220
$firstDayArray variable, 465
$name variable, 474
$newnum variable, 133
$SESSION superglobal, 229-232, 235
$start variable, 464
$txt variable, 135

Index
>=

(greater than or equal to) operators, 102, 316

<DirectoryMatch> directive container, 52

<Directory> directive container, 52

<FilesMatch> directive container, 52

<Files> directive container, 52

<IfDefine> conditional container, 53

<IfModule> conditional container, 53

<LocationMatch> directive container, 52

<Location> directive container, 52

<pre> tags, viewing multiple spaces in HTML, 175

<VirtualHost> directive container, 52

< (less than) operators, 102, 316

<= (less than or equal to) operators, 316

- (minus signs)
  field width specifiers, 176
  subtraction operators, 98
  -c command-line option, 486

-D httpd option, server binary, 57

-DMyModule switch, 53

-f httpd option, server binary, 57

-I httpd option, server binary, 57

-v httpd option, server binary, 57

% (percent signs)
  conversion specification, 172-173
  log formats, 502
  modulus operators, 98
  wildcards, 38
  %a format string option (DATE FORMATT() function), 342
  %A formatting directive, 502
  %a formatting directive, 502
  %b format string option (DATE FORMATT() function), 342
  %B formatting directive, 503
  %b formatting directive, 503
  %c format string option (DATE FORMATT() function), 342
  %C formatting directive, 503
  %D format string option (DATE FORMATT() function), 342
  %d formatting directive, 502
  %E format string option (DATE FORMATT() function), 342
  %e formatting directive, 502
  %F formatting directive, 503
  %H format string option (DATE FORMATT() function), 342
  %h formatting directive, 502-503
  %I format string option (DATE FORMATT() function), 343
  %i formatting directive, 503
  %J format string option (DATE FORMATT() function), 342
  %j formatting directive, 502-503
  %k format string option (DATE FORMATT() function), 342
  %K formatting directive, 503
  %l format string option (DATE FORMATT() function), 342
  %L formatting directive, 502
  %M format string option (DATE FORMATT() function), 342
  %m formatting directive, 503
  %n formatting directive, 503
  %p format string option (DATE FORMATT() function), 343
  %P formatting directive, 503
  %r format string option (DATE FORMATT() function), 343
  %R formatting directive, 503
  %s format string option (DATE FORMATT() function), 343
  %S formatting directive, 503
  %t format string option (DATE FORMATT() function), 342
  %T formatting directive, 503
  %T formatting directive, 502-503
  %u format string option (DATE FORMATT() function), 342
  %U formatting directive, 502-503
  %v format string option (DATE FORMATT() function), 342
  %V formatting directive, 502-503
  %W format string option (DATE FORMATT() function), 342
  %X formatting directive, 503
  %x format string option (DATE FORMATT() function), 342
  %Y formatting directive, 503
  || (or) operators, 102-103
  + (addition) operators, 97-98
  # (pound signs), 51, 69
  ? (ternary) operators, 116
  * (quotation marks), escaping in strings, 124
  ; (semicolons)
    instruction terminators, 88
    division operators, 98
    escaping quotation marks in strings, 124
    /* operators, 81
    // (forward slashes), 81
    /* operators, 81
    // (forward slashes), 81
    /tmp directory, 230
    /usr/local/apache2 directory, 47
    /usr/local/php/lib directory, 73
    /usr/local/src/ directory, 66
    /usr/src/ directory, 66
  
NUMBERS

3D pie charts, 277-278
Access control

- file-based control of, 486
- granting, 484
- limiting, HTTP methods, 491
- methods, combining, 490
- restricting, 488-490
  - authentication, 481-482
  - authentication modules, 484-487
  - based on cookie values, 493-496
- client authentication, 483
- rules, 488-489
  - all clients, 489
  - domain names, 488
  - environment variables, 489
  - evaluating, 489-490
- security, 491

Access denied messages, troubleshooting

- Apache startup, 61
- MySQL installations, 32

Access logs (Apache), 56

access reports, creating, 513-514

AccessFileName directive, per-directory configuration files, 55

ACTION argument, input forms, 202

AddCharset directive, 519

addentry.php script, 409-412

addentry.php record addition script, 394

AddLanguage directive, 519

addNums() function, 136

address books (online)
  - database tables, planning/creating, 389
  - email tables, 391
  - fax tables, 391
  - field names, 390
  - master name tables, 390
  - personal notes tables, 392
  - telephone tables, 391
  - include files, creating, 392
  - menus, creating, 393

addresses
  - IP addresses
    - control access rules, 488
    - reverse DNS lookups, 504
  - listening addresses (Listen directive), 59

addtocard.php script, 450-451

alerts, LogLevel directive option, 508

algorithms
  - digest algorithms, 482, 564
  - symmetric cryptography, 562

all clients, access control rules, 489

ALL command, 37

Allow directives, access control rules, 488

Allow, Deny arguments, Order directive, 490

AllowOverride directives, per-directory configuration files, 55

ALTER command, 37

and operators (&&), 103

Apache

- commands
  - control script, 58
  - server binary, 57-58
- configuration changes, internationalization efforts, 519
- configuring
  - conditional evaluations, 53
  - containers, 52-53
  - directives, 51-52
  - file structure, 50
  - per-directory files, 54-55
  - ServerRoot directive, 54
- DoS attacks, 551
- installing
  - binary installations, 44-45
  - current and future versions, 43-44
  - from source code, 44
  - Linux/UNIX installation, 8-9, 45-47
  - methods for, selecting, 44
  - Mac OS X installation, 47-48
  - PHP installation, 66-68
  - Windows installation, 13-14, 48-50
- licenses, 48
- log files, 56
- logs, managing, 509-511
- MPM scalability, 544
- performance
  - caching, 550
  - LimitRequestBody directives, 551
  - LimitRequestFields directives, 551
  - LimitRequestFieldSize directives, 551
Apache News and Announcements list, 591
Apache Software Foundation website, 44
apache.exe, 57-58
ApacheBench (ab) performance tool, 548-549
apachectl script, 53
apachectl tool, control script command (UNIX), 58
appending files, 249, 255
application localization
character sets, 518
environment modifications, configuration changes to
Apache, 519
MySQL, 521
PHP, 520
internationalization, 517
web page structure, 521-524
apxs utility, 10
args
ImageArc() function, 272
ImageFilledArc() function, 274
arguments, 132
ACTION, input forms, 202
AllowOverride directive, 55
CustomLog directive, 506
default values, setting, 142-143
directives, 51
ENCRYPT, 218
flock() function, 256
HostNameLookups directive, 504
LogFormat directive, 504
logs, rotating, 510
optional, setting as, 143
ServerRoot directive, 54
swapping, 179-180
syslog daemon errors, logging (UNIX), 508
TYPE, 218
variables, passing references to, 143-144
arithmetic functions, dates/times, 344-345
arithmetic operators, 98
array data types, 91
array() function, 92, 150-151
arrays
array keys() function, 155
array merge() function, 155
array operator, creating via, 150-151
array pop() function, 155
array push() function, 155
array shift() function, 155
array unshift() function, 155
array values() function, 155
associative arrays
creating, 151-152
getdate() function, 192
breaking strings into, 190-191
count() function, 154
defining, 149
each() function, 154
foreach() function, 154
HTML form input, accessing via, 203-204
index positions, creating, 150-151
list() function, 154
multidimensional arrays, creating, 152-153
mysqli fetch array() function, 370
reset() function, 154
session variables, adding to, 231
shuffle() function, 155
sizeof() function, 154
variables, 90
ASP tags, 77
asp tags setting, 77
assignment operators (=), 89, 97-99
asterisks (*)
multiplication operators, 98
table creation, 313
wildcards, 38
asymmetric cryptography. See public key cryptography
auth cookies, 496-497
auth users table, 493
AuthConfig, directive value, 55
AuthDBMGroupFile directive, database file-based access control authentication, 487
AuthDBMUserFile directive, database file-based access control authentication, 487
authentication, 561
  access, restricting, 481-482
  basic, 482
  browsers, AuthType directive, 483
  client authentication, restricting access, 483
  database file-based access control, 486-487
defined, 481
digest, 482-483
digital certificates, 565-566
file-based authentication, 484-486
  modules
    access (restricting), 484-487
directives, 483
  functions, 484
MySQL errors in, 36
process of, 35-36
realm authentication, AuthName directive, 483
SSL protocols, 564-567
AuthGroupFile directive, user file backend storage, 485
AuthName directive, authentication modules, 483
Authoritative directives, file-based authentication, 486
authorization, 481
AuthType directive, authentication modules, 483
AuthUserFile directive, user file backend storage, 485
awstats log analysis, 511

B

backend storage
database file-based access control authentication, 487
file-based authentication, 485
functions, authentication modules, 484
backslashes (\)
  converting time stamps to dates, 195
directives, 51
  escaping quotation marks in strings, 124
basic authentication, 482
benchmark() function, 576-577
BIGINT data type, 306
binary distribution, installing MySQL from, 23
binary installation, Apache, 44
BINARY keyword, 317
binary server commands, 57-58
bind to port, troubleshooting, 61
BLOB data type, 308
boolean data types, 91
breadcrumb trails, 443
break statements
case statements, 115
code readability, 124
loops, 121-122
browser authentication, AuthType directive, 483
browsers
  access, environment variables, 489
  Apache, accessing, 60
  cookies, viewing, 226
digest authentication, 483
built-in functions, 132

C

CA (Certification Authorities)
certificate signing requests, 572
digital certificates, 565
CacheFile directive, mapping files, 549
caching Apache performance, 550
calendar example
events, adding, 465-472
HTML form, 460-462
library, creating, 473-479
table, creating, 462-465
user input, 459-460
calling functions, 132-134, 140-142
Can’t connect to server messages, troubleshooting MySQL installations, 32
CAPTCHAs, 286-288
case (string text), converting, 187-188
case statements, 115
case-sensitivity, constant names, 105-106
casting variables, 94-95
cat id field, storefront database table example, 434
certificates
digital certificates
  authentication, SSL protocols, 565
CA (Certification Authorities), 565
  chaining, 565
  information, 566
SSL, 565-566
managing (secure servers), 570-572
self-signed (managing certificates), 572
signing requests, 571-572
CGI errors, logging, 507

How can we make this index more useful? Email us at indexes@samspublishing.com
changelog

changelog, software upgrades, 592
CHAR(M) data type, 308
Character Set Options screen (MySQL Configuration Wizard), 30
character sets, 518
CHARSET variable, 522
charts (pie), creating, 275-278
checkdate() function, 196, 459
ciphertext message encryption, 562
classes
  defined, 159
  properties, 163
clauses
  else, 465
  ON, 320
  ORDER BY, 313, 328, 338
  WHERE, 315-317
CLF (Common Log Format), 503
clients
  all clients, access control rules, 489
  authentication
    basic authentication, 482
    digest, 482-483
    restricting access, 483
    user management, 483
  tracking
    access logs, 56
    troubleshooting, 505
code
  blocks
    echo() statements, 126
    HTML code, returning to, 127
    iteration (loops), 118
  PHP, adding comments to, 80-81
  status code, conditional looping, 505
  when to comment, 82
collision resistant message digest algorithms, 564
  color
    fills, 274
    ImageColorAllocate() function, 272
    RGB color value, image creation process, 270
column command (UNIX), passing data to, 261
columns priv tables, 35
  combining
    access methods, 490
    assignment operators, 99
  command-line arguments, 66
  comments
    defined, 80
    multiline, 81
    PHP code, adding to, 80-81
    single-line, 81
    when to comment, 82
  COMMIT command, database transactions, 354-356
  Common Name field, certificate signing requests, 571
  communications, security, 561
  Compact installation option (MySQL Setup Wizard), 26
  comparison operators, 101-102
    case sensitivity in, 317
    equal to (=), 322
    WHERE clauses, 316-317
  compatibility schemas, 51
  compiling Apache
    installations, 47
    modules, 46
  compress command, 45-46
  compression
    reduced transmitted data (Apache performance), 550
    uncompressing source code (Apache installations), 45-46
  CONCAT WS() function, 330, 341
  CONCAT() function, 329-330
  concatenation functions, 329-330
  concatenation operators, 98, 214
  conditional DELETE statements, 327-328
  conditional evaluation, Apache configuration, 53
  conditional logging
    CustomLog directives, 506
    HostNameLookups directive, 504
    HTTP requests, 505-506
    IdentityCheck directive, 505
  conditional looping
    environment variables, 506
    status code, 505
  conditional statements, include() statements within, 242
  conditional UPDATE statements, 324
  confidentiality, SSL protocols, 561-563
  config script, installing OpenSSL libraries, 568
  config.log files, 47
  config.status files, 47
  configuration files, 55
    Apache, starting, 59
    conditional containers, 53
    Listen directive, 59
    modifying, 57
    MPM, processing, 53
    my-huge.cnf, 578
    my-large.cnf, 578
    my-medium.cnf, 578
    my-small.cnf, 578
    parameters, 59
    per-directory configuration files, file system access (scalability), 546
    ServerName directive, 59
  configure command, 8, 47
configure scripts, 67-68
Apache installation, 46-47
makefiles, 46
PHP installation, 66
targets, 46
configuring
Apache
conditional evaluation, 53
containers, 52-53
directives, 51-52
file structure, 50
per-directory files, 54-55
ServerRoot directive, 54
PHP 67-68
software (Apache installations), 46-47
SSL (secure servers), 572
systems for mail() function, 211-212
connection timeout variables, 588
connection status variables, 587
constants
accessing, 105
defining, 105
naming, 105-106
predefined, 106
constructors
defined, 474
objects, 164
containers
Apache configuration, 52-53
conditional containers, configuration files, 53
defined, 50
syntax, 53
VirtualHost container, IP-based virtual hosting, 553
content negotiation, file system
access (scalability), 546
content structure, XML, 530
context schema directives, 51
continue statements, 123-124
Control Apache command (Start menu), 60
control information (certificates), 565
control script commands (Apache-related), 58
conversion functions, date/time functions, 346-347
conversion specifiers
field width specifiers, 177-178
printf() function, 172-173
converting
string text case, 187-188
time stamps to dates
date() function, 193-195
getdate() function, 192-193
gmdate() function, 195
cookies
accessing, 224
auth cookies, 496-497
components of, 224
defining, 223
deleting, 227
domain field, 224
expiration dates, 224, 226
HTTP COOKIE variable, 225
path field, 224
printing, 225
Set-Cookie header, 224
setcookie() function, 225
size limits, 223
uses for, 223
values, restricting access based on, 493-496
viewing, 226
count() function, arrays, 154
CREATE command, 37
CREATE TABLE command, 309-310
cryptography
public key cryptography, 563
symmetric cryptography, 562-563
CURDATE() function, 346
CURRENT DATE() function, 346
CURRENT TIME() function, 346
CURRENT TIMESTAMP() function, 346
CURTIME() function, 346
Custom installation option (MySQL Setup Wizard), 26
custom installations, Apache (Windows), 50
custom logs, database tables, 511
code snippet, 512-513
sample reports, 513-515
CustomLog directives, 506
data types, 90
array, 91
boolean, 91
changing
by casting, 94-95
settype() function, 93
date/time, 308
defining, 306
double, 91-92
float, 91
integer, 91-92
NULL, 91
numeric, 306-307
object, 91
resource, 91
signed, 306
string, 91-92, 308-309
testing, 91-96
unsigned, 306
D

How can we make this index more useful? Email us at indexes@samspublishing.com
database tables

discussion forums, creating, 415-416
online address books, planning/creating for, 389
e-mail tables, 391
fax tables, 391
field names, 390
master name tables, 390
personal notes tables, 392
telephone tables, 391

databases

design process, overview of, 301-302
file-based access control authentication, 486-487
good design characteristics, 293-294
normalization, 293
  first normal form, 299
  flat tables, 298-299
  normal forms, 298
  second normal form, 300
  third normal form, 301
SHOW DATABASE command, 583
transactions, 353
  COMMIT command, 354-356
  examples of, 355-357
  ROLLBACK command, 354-356
  syntax of, 354
DATE ADD() function, 344-345
date added fields, shopping cart database tables, 446
DATE data type, 308
DATE FORMAT() function, 342-343, 347, 422
date pulldown libraries, 473
date pulldown.class php script, 478
date select() function, 477
DATE SUB() function, 344-345
date() function, 193-195, 246
date/time
calendar example, 459
  event additions, 465-466, 468-472
  HTML form, 460-462
  library, creating, 473-479
  table, creating, 462-465
  user input, 460
current dates/times, retrieving, 191
data types, 308
file validation, 246
functions, 342-343
  arithmetic, 344-345
  conversion, 346-347
days, 336-338
hours, 340-341
minutes, 340-341
months, 338-339
seconds, 340-341
special, 346-347
weeks, 339-340
years, 339
HH:MM:SS time format, 346
testing dates, 196
time stamps
  converting to dates, 192-195
  creating, 195
  defining, 191
UNIX epoch, 191
web resources, 197
YYYY-MM-DD date format, 346
DATETIME data type, 308
day functions, 336-338
DAYNAME() function, 338
DAYOFMONTH() function, 336-338
DAYOFWEEK() function, 336
DAYOFYEAR() function, 336-337
db tables, 35
dbmanange, database file-based access control authentication, 487
debug, LogLevel directive option, 509
DECIMAL (M,D) data type, 307
declaring
  functions, 134
  objects, 160
  variables, 89
    outside of functions, 138
    within functions, 137
decrementing, integer variables, 100-101
decryption, 562
default schema directives, 51
define() function, 105-106
defineStrings() function, 523-524
delentry.php script, 405-406
DELETE command, 36-40, 326, 365
  conditional DELETE statements, 327-328
  ORDER BY clause, 328
  subqueries, 322
deleting
  cookies, 227
  directories, 257
  files, 248
  HTML tags from strings, 185
  newlines from strings, 185
  tabs from strings, 185
  users, database file-based access control authentication, 487
  whitespace from strings, 185
deny directives, access control rules, 488
deny, Allow arguments, Order directive, 489
DES symmetric cryptography, 562
destroying sessions, 234
die() function, 249
digest algorithm, 482, 564
dig
digest authentication, 482-483
dig
digital certificates
authentication, SSL protocols, 565
CA, 565
chaining, 565
checking, 565
information, 566
SSL, 565-566
directives
AccessFileName directive, per-directory configuration files, 55
AllowOverride directive, per-directory configuration files, 55
Apache configuration, 51-52
arguments, 51
AuthConfig value, 55
authentication modules, 483
AuthName directive, authentication modules, 483
Authoritative directive, file-based authentication, 486
AuthType directive, authentication modules, 483
CacheFile directive, mapping files (memory), 549
containers, syntax, 53
defined, 50
directories, applying in, 52
FileInfo value, 55
files, applying in, 52
flag directives, 73
formatting directives, logging (HTTP requests), 502-503
identifiers, status codes, 505
Indexes value, 55
KeepAliveTimeout directive, network settings (Apache performance), 551
Limit value, 55
LimitRequestBody directive, Apache performance, 551
LimitRequestFields directive, Apache performance, 551
LimitRequestFieldSize directive, Apache performance, 551
LimitRequestLine directive, Apache performance, 551
LimitXMLRequestBody directive, Apache performance, 551
Listen directive, 59, 553
LoadModule directive, SSL configurations, 572
MMAPFile directive, mapping files (memory), 549
mod vhost alias directive (mass virtual hosting), 556
NameVirtualHost directive, 554
Options directive, 546, 556
Options value, 55
Order directive, control access rules (evaluating), 489
processing, 53
Require directive, authentication modules, 483
schemas, 51
ScoreBoardFile directive, 547
ScriptAlias directive (mass virtual hosting), 556
ServerAlias directive (syntax), 555
ServerName directive, configuration files, 59
ServerRoot directive, Apache configuration, 54
SSLCertificateFile directive, SSL configurations (certificates and keys), 573
TimeOut directive, Apache performance, 551
URL, applying in, 52
values, 55, 73
VirtualDocumentRoot directive (mass virtual hosting), 556
VirtualDocumentRootIP directive (mass virtual hosting), 556
VirtualScriptAlias directive (mass virtual hosting), 556
VirtualScriptAliasIP directive (mass virtual hosting), 556
directories
/tmp, 230
/usr/local/apach2, 47
/usr/local/php/lib, 73
/usr/local/src, 66
/usr/src/, 66
contents, reading, 258-259
deleting, 257
directives, applying, 52
htdocs subdirectory, 74
lib subdirectory, 73
listing, creating (UNIX), 262
opening, 257-258
per-directory files (Apache configuration), 54-55
validating, file/directory confirmation, 244
discussion forums
database tables, creating, 415-416
include files, creating, 416-417
input forms, creating, 417
input scripts, creating, 418-419
posts, adding to topics, 428-431
topic lists, displaying, 421-423
topic posts, displaying, 424-426
DISTINCT variable, 339
division operators (/), 98
DN (distinguished names), 566
DNS (domain name servers), 552
do…while statements, 119
DocumentRoot, virtual hosting, 553
documents
  included files, 239-240
  conditional statements, 242
  include once() function, 243
  include path directive, 243
  loops, 242
  return values, 241
text, formatting as, 175
doDB() function, 377, 380-381
DOM functions, accessing XML from PHP, 532-533
domain field (cookies), 224
domain names, access control rules, 488
DoS (Denial of Service) attacks, Apache, 551
DOUBLE (M,D) data type, 307
double data types, 91-92
downloading
  PHP distribution files, 71
  source code (Apache installations), 45
drawing images
  color fills, 274
  from existing images, 279-280
  ImageColorAllocate() function, 272
  ImageCreate() function, 271
  lines, 272-273
  pie charts, 275-278
  shapes, 272-273
  transparent, 281
  x-axis coordinates, 272
  y-axis coordinates, 272
DROP command, 37
drop-down list boxes, 283
DSO (Dynamic Shared Objects), 8

E
each() function
  arrays, 154
  multidimensional arrays, 153
echo statements, 78, 126, 153
echo() function, 132
editors
  avoiding for PHP code, 82
  HTML, 76
ellipses
  ImageEllipse() function, 272
  ImageFilledEllipse() function, 274
else clauses, 112-113, 465
else statements, 467
elseif clauses, 113-115
email
  php.ini files, 383
  sending feedback forms, 212-214, 216
  system configuration for, 211-212
  email fields, subscriber tables (mailing lists), 376
  email tables, online address books, 391
  emailChecker() function, 377, 380-381
emerg, LogLevel directive option, 508
ENCRIPT argument, 218
encryption
  decryption, 562
  passwords, user management (file-based authentication), 485
  SSL protocols, 562
encryption keys, 562
end tags, 77-78, 82
ENUM data type, 309
environment modifications (internationalization), configuration changes to
  Apache, 519
  MySQL, 521
  PHP, 520
equation variables
  access control rules, 489
  conditional logging, 506
equal sign (=)
  assignment operators, 89, 97
  concatenation operators, 214
  equal to operators, 316, 322
  equivalence operators (==), 102
  identical operators (===), 102
error messages
  Access denied, 32
  Can’t connect to server, 32
  mysql error() function, 365
errors
  logging, 507-509
  Apache log files, 56
  ErrorLog directive, 507
  files, 507
  LogLevel directive, 508-509
  monitoring, 511
  programs, 507
  syslog daemon (Unix), 508
MySQL authentication process, 36
escapeshellarg() function, 265
escapeshellcmd() function, 265
events
  adding to calendars, 465-472
  recording, error logs, 56
exclamation marks (!)
  nonequivalence (!=) operators, 102, 316
  not operators (!), 103
exec() function, 262-263


exit statements, 211
expiration dates, cookies, 224-226
EXPLAIN command, 580
explode() function, 190-191
expressions, 97

Facebook, XML uses in, 531-532
FAQ (Frequently Asked Questions), 75
fax tables, online address books, 391
fclose() function, 249
feedback forms
  creating, 212
  email, sending via, 213-214
  HTML, formatting via, 216
feof() function, 250
fgetc() function, 253
fgets() function, 250
field names versus strings, 341
field width specifiers, 175
  conversion specifiers, 177-178
  precision specifiers, 176
FILE command, 38
fileatime() function, 246
filectime() function, 246
FileInfo, directive value, 55
fget() function, 250
field names versus strings, 341
field width specifiers, 175
  conversion specifiers, 177-178
  precision specifiers, 176
FILE command, 38
fileatime() function, 246
filectime() function, 246
FileInfo, directive value, 55
fget() function, 250
files
  Apache file systems, 545-547
  appending, 249, 255
  authentication
    Authoritative directive, 486
    backend storage, 485
    mod auth module, 484-486
    user management, 485
  closing, 249
  config.log, 47
  config.status, 47
configuration files, 55
  Apache, starting in, 59
  conditional containers, 53
  Listen directive, 59
  modifying, 57
  parameters, 59
  processing (MPM), 53
  ServerName directive, 59
  ServerRoot directive, 54
  creating, 248
  deleting, 248
descriptors, operating system scalability limits, 544
directives, applying, 52
ers, logging, 507
executability, 245
group files, backend storage (file-based authentication), 485
HTTP requests, logging, 505
httpd, 72
httpd.conf file, 59, 68
  included files, 239-240
  conditional statements, 242
  include once() function, 243
  include path directive, 243
  loops, 242
  return values, 241
INSTALL, 23
locking, 256
log files
  access logs, 56
  error logs, 56, 507
  paths (logfile), 511
  scoreboard file, 56
  mapping to memory (Apache performance), 549
  names, defining (logging), 505
  navigating, fseek() function, 252-253
opening, 249
password files, storing (file-based authentication), 486
per-directory configuration files, 54-56, 546
php.ini files, 68, 73
phpinfo.php files, 74
reading, 245, 260
  feof() function, 250
  fgets() function, 250
  fread() function, 251-253
  fseek() function, 252-253
README, 23
robots.txt files, Apache security, 552
scoreboard files, 547
status, checking, 245
structure of, Apache configuration, 50
testing, 246-248
upload forms, 217-220
user files, backend storage (file-based authentication), 485
validating
  checking existence of, 244
date/time information, 246
determining file size, 245
file status, 245
file/directory confirmation, 244
testing functions, 246-248
writing to, 245, 255
filesize() function, 245
file_exists() function, 244
fills (color), 274
finding
  string lengths, 181
  substrings, 181-182
first normal forms, 298-299
flag directives, 73
flat tables, 298-299
float data types, 91, 307
flock() function, 256
flow control
code blocks
echo() statements, 126
HTML code, returning to, 127
iteration (loops), 118
loops
break statements, 121-124
continue statements, 123-124
do...while statements, 119
for statements, 120-121
infinite loops, 120
nesting loops, 124-125
while statements, 117-118
switching flow
if else statements, 112-113
if elseif statements, 113-115
if statements, 112
switch statements, 115-116
ternary (?=) operators, 116
FLUSH command, 38, 581-582
FLUSH HOSTS command, 582
FLUSH LOGS command, 582
FLUSH PRIVILEGES command, 39
FLUSH TABLES command, 582-584
FollowSymLinks parameter, Options directive, 546
fonts (text)
custom fonts, image creation, 287-288
imageloadfont() function, 283
imagettftext() function, 283
specification, 283
fontWrap() function, 143
fopen() function, 249, 255
for statements, 120-123
foreach loops, multidimensional arrays, 153
foreach statements, 220
foreach() function, 154
foreign languages. See application localization
format control strings, printf() function, 172
formatting
dates/times, 342-343
documents as text, 175
feedback forms via HTML, 216
logs, 502-504
strings
argument swapping, 179-180
field width specifiers, 175-178
printf() function, 172-175
storing, 180
times/dates, 342-343
forms (HTML)
feedback forms
creating, 212
formatting, 216
sending via email, 213-214
file upload forms, 217-220
input forms
accessing input via arrays, 203-204
creating, 201-202
PHP/HTML combination forms
hidden fields, 208-209
HTML form, calling itself, 206
PHP number-guessing scripts, 206-208
redirecting users, 209-211
server headers, 210
forums (discussion tables), 416
fputs() function, 255
FQDN (fully qualified domain name), 553, 570
tread() function, 251-253
FreeBSD, 45
From headers, 212-214
FROM UNIXTIME() function, 347
fseek() function, 252-253
FTP client, 76
function exists() function, 145
function statements, 133
functions, 131. See also methods
abs(), 133
addNums(), 136
arguments, 132
optional arguments, 143
passing variable references to, 143-144
setting default values, 142-143
array keys(), 155
array merge(), 155
array pop(), 155
array push(), 155
array shift(), 155
array unshift(), 155
array values(), 155
array(), 92, 150-151
built-in, 132
calling, 132-134, 140-142
csetattr(), 196, 459
CONCAT WS(), 330, 341
CONCAT(), 329-330
constructors and, 474
count(), 154
CURDATE(), 346
CURRENT DATE(), 346
CURRENT TIME(), 346
CURRENT TIMESTAMP(), 346
CURTIME(), 346
DATE ADD(), 344-345
DATE FORMAT(), 342-343, 347
date select(), 477
DATE SUB(), 344-345
date(), 193-195, 246
date/time, 342-343
arithmetic, 344-345
conversion, 346-347
days, 336-338
hours, 340-341
minutes, 340-341
months, 338-339
seconds, 340-341
special, 346-347
weeks, 339-340
years, 339
DAYNAME(), 338
DAYOFMONTH(), 336-338
DAYOFWEEK(), 336
DAYOFYEAR(), 336-337
declaring, 134
define(), 105-106
defined, 78, 132-133
die(), 249
doDB(), 380-381
DOM, accessing XML from PHP, 532-533
each(), 153-154
echo(), 132
eEmailChecker(), 380-381
escapeshellarg(), 265
escapeshellcmd(), 265
eexec(), 262-263
explode(), 190-191
fclose(), 249
feof(), 250
fgets(), 253
fgetss(), 250
filesize(), 245
file_exists(), 244
flock(), 256
fontWrap(), 143
fopen(), 249, 255
foreach(), 154
fputs(), 255
fread(), 251-253
FROM UNIXTIME(), 347
tseek(), 252-253
function exists(), 145
fwrite(), 255
getdate(), 192-193, 460, 464, 475
gettype(), 94, 97
getYearEnd(), 476
getYearStart(), 476
gmdate(), 195
header(), 175, 210-211
HOUR(), 340
include files, creating,
mailing list subscription
mechanisms, 376-377
online address books, 392
include_once(), 243
is dir(), 244
is file(), 244
is uploaded file(), 220
isset(), 459
is_executable(), 245
is_file(), 248
is_readable(), 245
is_writable(), 245
LCASE(), 334
LEADING, 332
LEFT(), 334
list(), 154
LOCATE(), 332
LPAD(), 332
LTRIM(), 331
ltrim(), 185
mail()
parameters of, 214
system configuration for,
211-212
MINUTE(), 340
mkdir(), 257
mktime(), 195, 460, 475
month select(), 477
MONTH(), 338
MONTHNAME(), 338
move uploaded file(), 220
MySQL, accessing list of, 371
mysqli *, 361
naming, 134-135
nl2br(), 189
NOW(), 346
numberedHeading(), 141-142
opendir(), 257-258
output(), 477
passthru(), 264
phpinfo(), 74
popen(), 260-261
printf(), 78-79, 132
printBR(), 135
println()
conversion specification,
172-173
format control strings, 172
padding specifiers, 174-175
type specifiers, 173
readdir(), 258-259
REPEAT(), 335
REPLACE(), 335
reset(), 154
RIGHT(), 334

How can we make this index more useful? Email us at indexes@samspublishing.com
rmdir(), 257
RPAD(), 332
rtrim(), 185
RTROM(), 331
SEC TO TIME(), 347
SECOND(), 340
serialize(), 231
session id(), 228
session save path(), 230
session set save handler(), 228
session start(), 228, 232, 447
session_destroy(), 234
setcookie(), 225-227
setDate array(), 475
setDate global(), 475-477
settype(), 93-95
setYearEnd(), 476
setYearStart(), 476
shuffle(), 155
SimpleXML, accessing XML from PHP, 535-537
sizeof(), 154
sprintf(), 180, 477
start session(), 229
str_replace(), 187
string
concatenation, 329-330
length, 329-330
location, 332
modification, 334-335
padding, 331-332
position, 332
substring, 333-334
trimming, 331-332
strip_tags(), 185
strlen(), 181
strpos(), 182
strstr(), 181-182
strtok(), 183
strtolower(), 188
 strtoupper(), 132, 188
substr(), 182-183
SUBSTRING(), 333
substr_replace(), 186
SYSDATE(), 346
system(), 263
tagWrap(), 145-146
test(), 138
testing, 145-146
TIME FORMAT(), 343
TIME TO SEC(), 347
time(), 191, 226
touch(), 248
TRAILING, 332
trim(), 185
UCASE(), 334
ucfirst(), 188
ucwords(), 188
underline(), 145-146
UNIX TIMESTAMP(), 347
unlink(), 248
user-defined, 133
values, returning, 136
variables
accessing globals via global statements, 139
changing globals within functions, 140
declaring outside of, 138
declaring within, 137
passing references to, 143-144
remembering values between calls, 140-142
scope of, 137
WEEKDAY(), 336
wordwrap(), 189-190
year select(), 477-478
YEAR(), 339

G
GD Library, 270
GET method, input forms, 203
getdate() function, 192-193, 460, 464, 475
gettype() function, 94, 97
getYearEnd() function, 476
getYearStart() function, 476
GIF images, logging, 506
giftopnm shell utility, 264
global statements
globals, accessing via, 139
variables, remembering values between function calls, 140
globals, 89
functions, changing within, 140
global statements, accessing via, 139
gmdate() function, 195
GRANT command, 37-39
granting
access, 484
privileges, 37-39
greater than operators (>), 102, 316
greater than or equal to operators (>=), 102, 316
group settings, troubleshooting
Apache startup, 61
groups file, backend storage, 485
gunzip command, 23, 45
gzip command, 45

H
hard drives, MySQL optimization, 576
hardware load balancer (Apache performance), 550
hash, defined, 493
header() function, 175, 210-211
headers
  From, 214
  Host header, name-based virtual hosting, 553-555
  HTTP headers, caching (Apache performance), 550
  messages, character sets, 518
  Reply-to, 214
  request headers, name-based virtual hosting, 554
  Set-Cookie, 224
help, PHP installation, 74-75
HH:MM:SS time format, 346
hidden fields (forms), 208-209
HMAC (Hash Message Authentication Code), 564
host tables, 35
HostnameLookups, network setting (scalability), 547
HostNameLookups directive, 489, 504
hostnames, resolving (managing logs), 509
hour functions, 340-341
HOUR() function, 340
.htaccess, per-directory configuration files, 546
htdocs subdirectory, 74
HTML (Hypertext Markup Language)
  calendar example, 460-462
  code blocks, 127
  editors, 76
  forms
    feedback forms, 212-216
    file upload forms, 217-220
    input forms, 201-204
    PHP/HTML combination forms, 206-211
    multiple spaces, viewing, 175
PHP combination, 79
tags, deleting from strings, 185
htpasswd utility, managing user password files, 485
HTTP (Hypertext Transfer Protocol)
  COOKIE variable, 225
  headers, caching (Apache performance), 550
  methods, access (limiting), 491
  requests, logging, 503-504
  conditional logging, 505-506
  files, 505
  programs, 506-507
  secure HTTP 562
httpd server binary command (Unix), 57
httpd.conf configuration file, 50, 59, 68, 72
httpd.pid file, 56
HUP signals, sending, 57
id field, 366
identical operators (===), 102
IdentityCheck directive, conditional logging, 505
if statements, 249, 465
  comparison operators, 101
  else clauses, 112-113
  elseif clauses, 113-115
if…else statements, 207
ImageArc() function, 272
ImageColorAllocate() function, 272
ImageCreate() function, 271, 287
ImageCreateFromGif() function, 279
ImageCreateFromJpg() function, 279
ImageCreateFromPng() function, 279
ImageDestroy() function, 273, 287
ImageEllipse() function, 272
ImageFilledArc() function, 274, 276
ImageFilledEllipse() function, 274
ImageFilledPolygon() function, 274
ImageFilledRectangle() function, 274
ImageGif() function, 273
ImageJpeg() function, 273
ImageLine() function, 272
ImageLoadFont() function, 283, 287
ImagePng() function, 273, 287
ImagePolygon() function, 272
ImageRectangle() function, 272
images
  creating
    custom fonts, 287-288
    custom text, 287-288
    from user input, 282-285
    JPEG libraries, 271
    PHP distribution, 270-271
    PNG libraries, 271
    RGB color values, 270
    via scripts, 286-287
    zlib libraries, 271
drawing
  color fills, 274
  from existing images, 279-280
  ImageColorAllocate() function, 272
  ImageCreate() function, 272
  lines, 272-273
  pie charts, 275-278
  shapes, 272-273
  transparent images, 281
  x-axis coordinates, 272
  y-axis coordinates, 272
logging, 506
reduced transmitted data (Apache performance), 550
stacking, 281

How can we make this index more useful? Email us at indexes@samspublishing.com
imagestring() function, 282, 285-287
imagettftext() function, 283, 287
include files
   creating (mailing list subscription mechanisms), 376-377
discussion forums, creating for, 416-417
online address books, creating for, 392
include once() function, 243
include path directive, 243
include() statements, 239-241
   include once() function, 243
   include path directive, 243
   within conditional statements, 242
   within loops, 242
included files, 239-241
   conditional statements, 242
   include once() function, 243
   include path directive, 243
   loops, 242
   return values, 241
incrementing integer variables, 100-101
INDEX command, 38
index strings, 180-181
Indexes, directive value, 55
infinite loops, 120
inheritance, objects, 164-165
INNER JOIN command, 320
input forms
   accessing input via arrays, 203-204
creating, 201-202
discussion forums, creating for, 417
input scripts, creating for discussion forums, 418-419
insert Record button, 367
INSERT statement, 365
INSTALL file, 23
installing
   Apache
      binary installation, 44
current and future versions, 43-44
custom installation, 50
from source code, 44
Linux/UNIX installation, 8-9, 45-47
Mac OS X installation, 47-48
methods for, selecting, 44
typical installation, 50
Windows installation, 13-14, 48-50
MySQL
   current and future version information, 21-22
Linux/UNIX installation, 7, 22-23
Mac OS X installation, 17, 24
troubleshooting, 32
Windows installation, 11, 17, 26-27, 30-31
OpenSSL libraries
   SSL installations, 567
UNIX, 568-569
Windows, 567
PHP
   current and future versions, 65-66
   help for, 74-75
Linux/UNIX installation, 9-10, 16, 66-68
Mac OS X installation, 18-20, 69-70
testing, 74
Windows installation, 15-20, 71-72
SSL
   mod ssl module, 568-569
   OpenSSL library, 567
instruction terminators, 88
INT data type, 306
integer data types, 91-92
integer variables, incrementing/decrementing, 100-101
integrity
   communications security, 561
digest algorithms, 564
message digests, 564
internationalization, 517
   environment modifications, configuration changes to
      Apache, 519
      MySQL, 521
      PHP 520
   key aspects, 517
IP addresses
   control access rules, 488
   reverse DNS lookups, 504
IP-based virtual hosting, 552-553
   is dir() function, 244
   is file() function, 244
   is uploaded file() function, 220
   isset() function, 459
issuer information (certificates), 565
   is_executable() function, 245
   is_file() function, 248
   is_readable() function, 245
   is_writeable() function, 245
iterations (loops), 118, 123
leading spaces, padding specifiers, 175
LEFT JOIN command, 320-321
LEFT() function, 334
length functions, 329-330
less than operators (<), 102, 316
less than or equal to operators (<=), 102, 316
lib subdirectory, 73
libraries
calendar example, 473
GD, 270
JPEG, image creation, 271
OpenSSL, 567-569
PNG, image creation, 271
SSLeay, 567
zlib, image creation, 271
licenses (Apache), 48
LIKE operator, 317
LIMIT command, 314-315
Limit containers, HTTP methods, 491
LimitExcept containers, HTTP methods, 491
LimitRequestBody directive, Apache performance, 551
LimitRequestFields directive, Apache performance, 551
LimitRequestFieldSize directive, Apache performance, 551
LimitRequestLine directive, Apache performance, 551
LimitXMLRequestBody directive, Apache performance, 551
lines
drawing, 272-273
ImageLine() function, 272
Linux
Apache installation, 8-9, 60
configure script, 46-47
make command, 47
source code, downloading, 45
source code, uncompressing, 45-46
distribution CDs, 22
mod_ssl Apache modules, 569
MySQL installations, 7, 22-23
OpenSSL library, installing, 568
PHP installations, 9-10, 16, 66-70
server processes, operating system scalability limits, 544
list() function, arrays, 154
Listen directive, 59, 553
listening addresses (Listen directive), 59
listings
abs() function, 133
access reports, creating, 513-514
calendar
date pulldown.class php script, 478
display script, 462-464
event additions, 468-472
HTML form, 460-461
library, creating, 474-479
user input, checking, 460
viewing events, 470-472
objects
inheritance, 164-165
methods, 163-164
PHP script with HTML, 79
storefront database table example, 437, 441
transparent images, 281
user login script, 495
lists (user), Require directive, 483
In command, symlinks, 546
load distribution (Apache performance), 550
load testing, ApacheBench (ab) performance tool, 548-549

How can we make this index more useful? Email us at indexes@samspublishing.com
LoadModule directive

LoadModule directive, SSL configurations, 572
local variables, 89
local7 syslog daemon, logging errors, 508
locales, defining, 517
localization, 517
character sets, 518
environment modifications
Apache, configuration changes to, 519
MySQL, configuration changes to, 521
PHP, configuration changes to, 520
internationalization, 517
Web page structure, 521-524
LOCATE() function, 332
location functions, 332
lock screen mechanism, 34
locking files, 256
LogFormat directive, 504
logical operators, 102-103, 316
login forms, 494-496
LogLevel directive, 508-509
logname paths, log files, 511
log resolve utility, resolving hostnames, 509
logs
analyzing, 510-511
CLF (Common Log Format), 503
conditional logs
CustomLog directive, 506
HostNameLookups directive, 504
HTTP requests, 505-506
IdentityCheck directive, 505
custom logs, 511
database tables
code snippet, 512-513
sample reports, 513-515
database tables, creating, 511
directives, status codes, 505
errors, 507-509
files, 507
LogLevel directive, 508-509
monitoring, 511
programs, 507
syslog daemon (UNIX), 508
files
Apache, 56
paths, lognames, 511
formatting directives (HTTP requests), 502-504
HTTP requests, 501-504
files, 505
programs, 506-507
images, 506
managing
analysis, 510-511
Apache, 509-511
error logs (monitoring), 511
hostname resolution, 509
log rotation, 509-510
merging, 510
request logs, creating, 501
rotating, 509-510
splitting, 510
Logscan, monitoring error logs, 511
Logtools, log manipulation tools, 510
LONGBLOB data type, 309
LONGTEXT data type, 309
loops, 117
break statements, 121-122
conditional loops, 505-506
continue statements, 123
do...while statements, 119
for statements, 120-121
foreach loops, multidimensional arrays, 153
include() statements within, 242
infinite loops, 120
iteration, 118
iterations, skipping, 123
nesting loops, 124-125
while loops, 261, 426
while statements, 117-118
LPAD() function, 332
LTRIM() function, 331
ltrim() function, cleaning up strings, 185

M

MAC (message authentication codes), 564
SSL protocols, 564
Mac OS X
Apache installation, 47-48
MySQL installation, 17, 24
PHP installation, 18-20, 69-70
mail() function, 385
parameters of, 214
system configuration for, 211-212
mailing lists
mailing mechanisms, 383-385
MySQL, 33
PHP, 75
subscription mechanism, 375
include files, creating, 376-377
subscriber tables, creating, 376
subscription forms, creating, 377-380, 383
maintenance releases (software upgrades), 592
make command, 10, 67
make install command, 67
Apache installing, 47
PHP installation, 10
make utility, 47
makefiles, script configuration, 46
managing
 certificates (secure server)
    certificate signing requests
    creating, 571-572
    key pairs (creating), 570-571
    self-signed certificates, 572
logs
    analysis, 510-511
    Apache, 509-511
    error logs (monitoring), 511
    hostnames (resolving), 509
    log rotation, 509-510
    merging, 510
users
    database file-based access
        control authentication, 487
    file-based authentication, 485
many to one mappings, DNS virtual hosting, 552
many to many table relationships, 296-297
mapping files to memory (Apache performance), 549
mass virtual hosting, 555-556
master name tables, online address books, 390, 400-401
max connections variable, 588
MAX FILE SIZE field, file upload forms, 218
max used connections status variable, 587
mbstring related functions, 520
MD5 digest algorithms, 564
MEDIUMBLOB data type, 309
MEDIUMTEXT data type, 309
memory
    files, mapping to (Apache performance), 549
    MySQL optimization tips, 576
menus (online address books), creating, 393
merging logs, 510
message digest algorithms, 564
META tags, header messages, 519
methods. See also functions
    access methods, combining, 490
        defined, 159
    GET method, input forms, 203
        objects, 162-164
    POST method, input forms, 202-203
minimal installations, MySQL on Linux/UNIX, 23
minus signs (-)
    -c command-line option, 486
    -D httpd option, server binary, 57
    -DMyModule switch, 53
    -f httpd option, server binary, 57
    -l httpd option, server binary, 57
    -v httpd option, server binary, 57
    field width specifiers, 176
    subtraction operators, 98
minute functions, 340-341
    MINUTE() function, 340
mkdir() function, 257
mktimed() function, 195, 460, 475
MMAPFile directive, memory mapping files, 549
mod access module, access control, 488-489
mod auth dbm module, database file-based access control authentication, 486-487
mod auth module
    file-based authentication, 484-486
    sample configuration, 486
mod cache module, caching (Apache performance), 550
mod deflate module, reduced transmitted data (Apache performance), 551
mod file cache module, mapping files, 549
mod so command, 46
mod ssl module, SSL
    configurations, 572
    installations, 567-569
mod status module, network setting (scalability), 547
mod vhost alias directive (mass virtual hosting), 556
modification functions, 334-335
modifying
    configuration files, 57
    httpd.conf file, 59
modules
    Apache compiles, 46
    authentication
        access (restricting), 484-487
        directives, 483
        functions, 484
    mod access module, access control, 488-489
    mod auth, file-based authentication, 484-486
    mod auth dbm, database file-based access control authentication, 486-487
    mod auth module, sample configuration, 486
    mod cache, caching (Apache performance), 550

How can we make this index more useful? Email us at indexes@samspublishing.com
modules

mod deflate, reduced transmitted data (Apache performance), 551
mod file cache, files (mapping), 549
mod ssl, SSL configurations, 572 installations, 567-569
mod status, network setting (scalability), 547
schema directives, 51
modulus operators (%), 98
monitoring error logs, 511
month functions, 338-339
month select() function, 477
MONTH() function, 338
MONTHNAME() function, 338
move uploaded file() function, 220
MPM (Multi-Processing Module) configuration files, processing, 53 operating system scalability limits, 544
multibyte character sets, 518
multidimensional arrays, creating, 152-153
multiline comments, 81
multiplication operators (*), 98
Mutual-Failure arguments, Order directive, 490
my-huge.cnf configuration file, 578
my-large.cnf configuration file, 578
my-medium.cnf configuration file, 578
my-small.cnf configuration file, 578
MySQL, 22
Announcements list, 591 configuration changes, internationalization efforts, 521 connections, securing, 34
data inserting with PHP, 365-367 retrieving with PHP, 369-370

data types
date/time, 308 defining, 306 numeric, 306-307 signed, 306 string, 308-309 unsigned, 306

functions
accessing list of, 371 date/time, 336-347 string, 329-335
GUI administration tool, 34
installing
current and future version information, 21-22 Linux/UNIX installation, 7, 22-23 Mac OS X, 24 troubleshooting, 32 Windows installation, 11, 17, 26-27, 30-31
mailing lists, 33
optimization, improving
benchmark() function, 576-577
FLUSH command, 581-582
FLUSH HOSTS PRIVILEGES command, 582
FLUSH LOGS PRIVILEGES command, 582
FLUSH PRIVILEGES command, 581
FLUSH TABLES command, 584
FLUSH TABLES PRIVILEGES command, 582
OPTIMIZE TABLE command, 579
queries, 580
SHOW COLUMNS command, 585
SHOW command, 582
SHOW CREATE TABLE command, 584-585
SHOW DATABASES command, 583
SHOW GRANTS command, 583
SHOW INDEX command, 585
SHOW OPEN TABLES command, 584
SHOW STATUS command, 586
SHOW TABLES command, 584
SHOW VARIABLES command, 588

tips for, 575-576
PHP connections, 361
error messages, retrieving, 365
errors, 363
queries, executing, 363-364
syntax of, 362

privilege systems
authentication process, 35-36 columns priv tables, 35
db tables, 35 granting, 37-39
host tables, 35 overview, 35
revoking, 39

root users, running as, 33, 40
starting, 33-34

startup options, 577-579

support contracts, 33
upgrading, 593

web address, 22
MySQL Configuration Wizard, 27, 30-31
N

name-based virtual hosting, 552-555
NameVirtualHost directive, 554
naming
   constants, 105-106
   domains, access control rules, 488
   error log files, 507
   functions, 134-135
   logging files, 505
   uploaded files, 220
   variables, 88
navigating files via fseek() function, 252-253
negative terms, 73
nesting loops, 124-125
network settings
   Apache performance, 551
   scalability, 547
network/mask pairs, control access rules, 488

Networking Options screen (MySQL Configuration Wizard), 30
newline character (\n), 124, 214, 250
newlines, removing from strings, 185
NIS (Network Information Services), client authentication, 483
nl2br() function, 189
nonequivalence operators (!=), 102, 316
normal forms, 298
   first normal form rules, 299
   second normal form rules, 300
   third normal form rules, 301
normalization, 293
   defining, 298
   flat tables, 298-299
   normal forms, 298-301
   redundancy, 299
not operators (!), 103
NOW() function, 346, 419
NULL data types, 91
numberedHeading() function, 141-142
numeric data types, 306-307

O

objects
   constructors, 164
   creating, 159
      instances of, 160-161
      methods, 163-164
   data types, 91
   declaring, 160
   inheritance, 164-165
   methods, 162-163
   properties
      changing, 162
      viewing, 161
ON clause, 320
one to many mappings, DNS virtual hosting, 552
one to one mappings, DNS virtual hosting, 552
one to many table relationships, 296
one to one table relationships, 295
online address books
database tables, planning/creating, 389
   email tables, 391
   fax tables, 391
   field names, 390
   master name tables, 390
   personal notes tables, 392
   telephone tables, 391
   include files, creating, 392
   menus, creating, 393
   records
      adding subentries to, 407-412
      record addition mechanism, 394-397
      record deletion mechanism, 405-406
      viewing, 398-405
online storefront database table example
   cat id field, 434
   category of items, displaying, 437-440
   planning process, 433-434
   store categories field, 434-435
   store item color field, 434-437
   store item size table, 436
   store items field, 434-436
opendir() function, 257-258
openssl command-line tool (certificates), 570
OpenSSL libraries, 567-569
operands, 97
operators

&& (and) operators, 103
* (multiplication) operators, 98
*/ operators, 81
= (equal sign) assignment operators, 89, 97
comparison operators, 322
concatenation operators, 214
== (equivalence) operators, 102
=== (identical) operators, 102
! (not operators), 103
!= (nonequivalence) operators, 102
!=(not equal to) operators, 316
> (greater than) operators, 102, 316
>= (greater than or equal to) operators, 102, 316
<= (less than or equal to) operators, 316
- subtraction operators, 98
% modulus operators, 98
. concatenation operators, 98
? (ternary) operators, 116
arithmetic operators, 98
assignment operators, 97-99
comparison operators, 101-102
case sensitivity in, 317
equal to (=), 322
WHERE clauses, 316-317
concatenation operators, 98, 214
defining, 97
logical operators, 102-103, 316
operands, 97
post-decrement operators, 100
post-increment operators, 100
precedence of, 103-105

optimization, MySQL, 575
benchmark() function, 576-577
CPU, 576
database and table information retrieval, 583-584
FLUSH command, 581-582
hard drives, 576
memory, 576
operating systems, 576
queries, 580
SHOW command, 582-583
startup options, 577-579
system status retrieval, 587-588
table structure, 579, 584-586

OPTIMIZE TABLE command, 579

optional arguments in functions, 143
Options directive 545, 546
or die() constructs, 250
or operators (||), 102-103
ORDER BY clauses, 313, 328, 338
Order directives
- Allow, Deny argument, 490
control access rules, evaluating, 489
- Deny, Allow argument, 489
- Mutual-Failure argument, 490
OS (Operating Systems)
MySQL optimization tips, 576
scalability
- external process control, 545
- file descriptors, 544
- server processes, 544

output() function, 477
override schema directives, 51
ownership, verifying, 33

P

padding functions, 331-332
padding specifiers
- leading spaces, 175
- print() function, 174-175
- special characters in, 175
padlock icon, 565
pass phrases, creating key pairs, 570
passthru() function, 264
password files, storing (file-based authentication), 486
password() function, 493

passwords
- basic authentication, 482
digest authentication, 482
- encrypting, user management (file-based authentication), 485

paths, log files, 511

PEAR (PHP Extension and Application Repository), PHP extensions, 595
PECL (PHP Extension Community Library), 595

per-directory files
- Apache configuration, 54-55
- configuration files, file system access (scalability), 546

percent signs (%)
- %a format string option (DATE FORMAT() function), 342
- %a formatting directive, 502
- %b format string option (DATE FORMAT() function), 342
- %b formatting directive, 503
- %c format string option (DATE FORMAT() function), 342
- %C formatting directive, 503
- %D format string option (DATE FORMAT() function), 342
%D formatting directive, 502
%e format string option (DATE
FORMAT() function), 342
%e formatting directive, 502
%f formatting directive, 503
%H format string option (DATE
FORMAT() function), 342
%h formatting directive, 502-503
%i format string option (DATE FOR-
MAT() function), 343
%i formatting directive, 503
%j format string option (DATE FOR-
MAT() function), 342
%k format string option (DATE FOR-
MAT() function), 342
%l formatting directive, 502
%M format string option (DATE
FORMAT() function), 342
%m formatting directive, 503
%o formatting directive, 503
%p format string option (DATE
FORMAT() function), 343
%q formatting directive, 503
%r format string option (DATE
FORMAT() function), 343
%r formatting directive, 503
%s format string option (DATE
FORMAT() function), 343
%t format string option (DATE
FORMAT() function), 343
%T formatting directive, 502-503
%U format string option (DATE
FORMAT() function), 342
%V format string option (DATE
FORMAT() function), 342
%X format string option (DATE
FORMAT() function), 342
%X formatting directive, 503
%y format string option (DATE
FORMAT() function), 342
%y formatting directive, 503
conversion specification, 172-173
log formats, 502
modulus operators, 98
wildcards, 38

**performance, Apache**
caching, 550
LimitRequestBody directives, 551
LimitRequestFields directives,
551
LimitRequestFieldSize directives,
551
LimitRequestBody directives,
551
load distribution, 550
mapping files to memory, 549
network settings, 551
reduction of transmitted data,
550
scalability
file system access, 546
network and status settings,
547
TimeOut directives, 551

**period (.) concatenation operators, 98**

**permissions, 32. See also privileges, 35**

**personal notes tables (online address
books), 392, 402**

**PHP**
Announcements list, 591
Apache, integrating with
Linux/UNIX, 68-70
Windows, 72-73

ASP tags, 77
coding, adding comments to, 80-81
configuring, 67-68, 520
cookies, deleting, 227
delimiter tags, 77
distribution, image creation,
270-271
documentation, 218
HTML, combining with, 79
included files, 239-241
installing
current and future versions,
65-66
help for, 74-75
Linux/UNIX installation, 9-10,
16
on Linux/UNIX with Apache,
66-68
on Mac OS X, 69-70
on Windows, 71-72
testing, 74
Windows installation, 15-20
instruction terminators, 88
mailing lists, 75
MySQL connections, 361
error messages, retrieving,
365
errors, 363
queries, executing, 363-364
syntax of, 362
MySQL data, inserting with,
365-367
php.ini file, 73
retrieving MySQL data with,
369-370
Script tags, 77
scripts, 76
short tags, 77
standard tags, 77
PHP

start tags, 77
upgrading, 595
website, 66, 71, 75
XML
  DOM functions, 532-533
  SimpleXML functions, 535-537
  uses of, 531-532
PHP Manual website, 371
php.ini files, 68, 73, 383
PHP/HTML combination forms
  hidden fields, 208-209
  HTML form, calling itself, 206
  PHP number-guessing scripts, 206-208
  redirecting users, 209-211
  server headers, 210
phpinfo() function, 74
phpinfo.php file, 74
phyMyAdmin interface, 34
pie charts, creating, 275, 277-278
pipes, opening, 260
plaintext messages, encryption, 562
PNG libraries, image creation, 271
pnmscale shell utility, 264
polygons
  ImageFilledPolygon() function, 274
  ImagePolygon() function, 272
popen() function, 260-261
port connections variable, 588
port values (Listen directive), 59
ports, troubleshooting bind to port, 61
position functions, 332
positive terms, 73
POST method, input forms, 202-203
post-decrement operators, 100
post-increment operators, 100
posts, adding to discussion forum topics, 428-431
pound sign (#), 51, 69
ppmtogif shell utility, 264
precision specifiers (field width specifiers), 176
predefined constants, 106
printf() function, 78-79, 132
printBR() function, 135
printf() function
  conversion specification, 172-173
  format control strings, 172
  padding specifiers, 174-175
  type specifiers, 173
printing cookies, 225
privileges
  authentication process, 35-36
  columns priv tables, 35
  db tables, 35
  granting, 37-39
  host tables, 35
  MySQL, overview of, 35
  revoking, 39
  tables priv tables, 35
  user tables, 35
problems, MySQL installation, 33
procedures (stored)
  benefits of, 357
  syntax of, 358-359
PROCESS command, 38
programs
  error logging, 507
  HTTP requests, logging, 506-507
  rotatelogs, 510
prologs, XML, 529
properties
  classes, 163
  objects
    changing in, 162
    viewing, 161
ps command, 33
public key cryptography, SSL protocols, 563
public key information (certificates), 565

Q - R

queries
  executing, 363-364
  optimizing, 580
  subqueries, 322
query strings, passing in session ID, 233
question marks (?), ternary operators, 116
quotation marks ("), escaping in strings, 124

r (read) mode, 249
RAM disks, scoreboard files, 547
RC2 symmetric cryptography, 562
RC4 symmetric cryptography, 562
read (r) mode, 249
readdir() function, 258-259
reading
  directory contents, 258-259
files, 260
  feof() function, 250
  fgetc() function, 253
  fgets() function, 250
  fread() function, 251-253
  fseek() function, 252-253
who command output (UNIX), 260
README files, 23
recording events, error log, 56
records

adding subentries to the online address books, 407-412
adding subentries to in, 407-412
record addition mechanism, 394-397
record deletion mechanism, 405-406
viewing in, 398-405

rectangles

ImageFilledRectangle() function, 274
ImageRectangle() function, 272

redirecting users in HTML/PHP combination forms, 209-211

redundancy, normalization, 299

registered user sessions, 235
RELOAD command, 38
require_once() statements, 244
require() statements, 244
reset() function, arrays, 154
replacing
portions of strings, 186
string portions, 186
substrings, 187

Reply-to header, 214
relytopost.php script, 428-431
request headers, name-based virtual hosting (syntax), 554

requests. See also HTTP requests

client requests, tracking (access log), 56
logs, creating, 501
Require directive, authentication modules, 483
require once() statements, 244
require() statements, 244
reset() function, arrays, 154
restoring data types, 91
restricting access, 488-490
authentication, 481-487
based on cookie values, 493-496
client authentication, 483
resolving hostnames (managing logs), 509
resource data types, 91

redirecting users in HTML/PHP combination forms, 209-211

redundancy, normalization, 299
registered user sessions, 235
RELOAD command, 38
removefromcart.php script, 454
removing
directories, 257
files, 248
 privileges, 39
session variables, 234

REPEAT() function, 335
REPLACE command, 325-326
REPLACE() function, 335
replacing
portions of strings, 186
string portions, 186
substrings, 187
Reply-to header, 214
relytopost.php script, 428-431
request headers, name-based virtual hosting (syntax), 554

requests. See also HTTP requests

client requests, tracking (access log), 56
logs, creating, 501
Require directive, authentication modules, 483
require once() statements, 244
require() statements, 244
reset() function, arrays, 154
replacing
portions of strings, 186
string portions, 186
substrings, 187
Reply-to header, 214
relytopost.php script, 428-431
request headers, name-based virtual hosting (syntax), 554

requests. See also HTTP requests

client requests, tracking (access log), 56
logs, creating, 501
Require directive, authentication modules, 483
require once() statements, 244
require() statements, 244
reset() function, arrays, 154
replacing
portions of strings, 186
string portions, 186
substrings, 187
Reply-to header, 214
relytopost.php script, 428-431
request headers, name-based virtual hosting (syntax), 554

requests. See also HTTP requests

client requests, tracking (access log), 56
logs, creating, 501
Require directive, authentication modules, 483
require once() statements, 244
require() statements, 244
reset() function, arrays, 154
resolving hostnames (managing logs), 509
resource data types, 91
restricting access, 488-490
authentication, 481-487
based on cookie values, 493-496
client authentication, 483
resuming sessions, 228-229
removefromcart.php script, 454
removing
directories, 257
files, 248
 privileges, 39
session variables, 234

REPEAT() function, 335
REPLACE command, 325-326
REPLACE() function, 335
replacing
portions of strings, 186
string portions, 186
substrings, 187
Reply-to header, 214
relytopost.php script, 428-431
request headers, name-based virtual hosting (syntax), 554

requests. See also HTTP requests

client requests, tracking (access log), 56
logs, creating, 501
Require directive, authentication modules, 483
require once() statements, 244
require() statements, 244
reset() function, arrays, 154
resolving hostnames (managing logs), 509
resource data types, 91
restricting access, 488-490
authentication, 481-487
based on cookie values, 493-496
client authentication, 483
resuming sessions, 228-229
return statements, 136, 179
reverse DNS lookups, IP addresses, 504
REVOKE command, 39-40
RGB color values, image creation, 270
RIGHT JOIN command, 321
RIGHT() function, 334
RLimitCPU directive, 545
RLimitMem directive, 545
RLimitNProc directive, 545
rmdir() function, 257
robots.txt files, Apache security, 552
ROLLBACK command, database transactions, 354-356
root elements, XML, 530
root users, 33, 40
rotatelogs utility, 507, 510
round robin DNS, 552
rows, mysql num rows() function, 369
RPAD() function, 332
RTRIM() function, 331
rtrim() function, cleaning up strings, 185

S

Satisfy all directive, combining access methods, 490
Satisfy any directive, combining access methods, 490
Satisfy directive, combining access methods, 490
saving state via hidden fields, 208-209
sayHello() function, 164

scarcity, 543

Apache network settings, 547
Apache performance-related settings
file system access, 545-546
network/status settings, 547
Apache status settings, 547
operating system limits
external process control, 545
file descriptors, 544
server processes, 544
performance-related settings
file system access, 546
network and status settings, 547
ScanErrLog programs, monitoring error logs, 511
schemas, directives, 51
scoreboard files, 56, 547
screensavers, 34
Script tags, 77
ScriptAlias directive (mass virtual hosting), 556
scripts
addentry.php, 409-412
addtocart.php, 450-451
apachectl, 53

How can we make this index more useful? Email us at indexes@samspublishing.com
configure scripts, 67-68
makefiles, 46
OpenSSL library installation, 568
PHP installation, 66
targets, 46
date pulldown.class.php, 478
delentry.php, 405-406
file upload forms, 219-220
images, creating from, 286-287
input scripts, creating for discussion forums, 418-419
PHP, 76
removefromcart.php, 454
replytopost.php (discussion forums), 428-431
selentry.php, 398-405
showcart.php, 452-453
showitem.php, 448-449
split-file Perl, splitting logs, 510
topic list script (discussion forums), 421-422
topic post script (discussion forums), 424, 426
user login, 494-495
SEC TO TIME() function, 347
second functions, 340-341
second normal forms, rules for, 300
SECOND() function, 340
secure HTTP, 562
secure servers
  certificates, managing, 570-572
  SSL
    configuring, 572
    protocols, 562-567
security
  access control, 491
  Apache, 551
authentication
  digital certificates, 565-566
discussed, 561
need for, 564
basic authentication, 482
certificates
  key pairs, 570-571
  self-signed, 572
  signing requests, 571-572
communications, integrity, 561
confidentiality, 561
  public key cryptography, 563
  SSL protocols, 562
  symmetric cryptography, 562-563
digest authentication, 482
encryption, 562
integrity
  communications, 561
digest algorithms, 564
  message digests, 564
lock screen mechanism, 34
MySQL
  connections, securing, 34
  server startup procedures, 33-34
  need for, 561
reverse DNS lookups, 504
software upgrades, 592
SSH, 34
SSL
  configuration, 572-573
  mod ssl Apache Module, 568-569
  OpenSSL, 567-568
  protocols (secure servers), 562
  symlinks, 546
TLS (Transport Layer Security), 562
Security Options screen (MySQL Configuration Wizard), 30
sel * fields, shopping cart database tables, 446-447
sel item price fields, shopping cart database tables, 447
SELECT command, 36-38, 203-204, 312-314, 317-319, 322, 365
SELECT element, 462
selentry.php script, 398-405
self-signed certificates (managing certificates), 572
semicolons (;), 79
  instruction terminators, 88
sending
  email
    feedback forms, 212-216
    system configuration for, 211-212
  signals, kill command, 57
serialize() function, 231
ServerAlias directive (syntax), 555
ServerName directive, configuration files, 59
ServerRoot directive, Apache configuration, 54
servers
  binary commands, 57-58
  headers (HTML/PHP combination forms), 210
  loads, distributing (Apache performance), 550
  starting, troubleshooting, 573
  virtual servers, specifying (<VirtualHost> directive container), 52
  Web servers, Apache installations (Windows), 48
Service icon, 58
session id fields, shopping cart database tables, 446
session id() function, 228
session save path() function, 230
session set save handler() function, 228
session start() function, 228, 232, 447, 522

sessions
- destroying, 234
- functions, 227
  - session id(), 228
  - session save path(), 230
  - session set save handler(), 228
  - session start(), 228, 232
  - start session(), 229

ID, passing in query strings, 233
registered users, 235
resuming, 228-229
starting, 228-229
state, storing, 228
user preferences, 235

variables
- accessing, 232
- accessing stored variables, 229-231
- adding arrays to, 231
- removing, 234
- storing, 229

session_destroy() function, 234

SET data type, 309
set time limit() function, 385
Set-Cookie header, 224
setcookie() function, 225-227
setDate array() function, 475
setDate global() function, 475-477
setName() function, 164
settype() function, 93, 95
setYearEnd() function, 476
setYearStart() function, 476
SHA, digest algorithms, 564

shading effects, pie charts, 277-278
shapes, drawing, 272-273
shopping cart database table example
carts
  - adding items to, 450-451
  - removing items from, 454-455
viewing, 452-454
checkout actions, performing, 456-457
checkout form, creating, 456
field lengths, 446
field names, 445-446
integrating with storefront, 447-449
short open tag switches, 77
short tags, 77

SHOW COLUMNS command, 585
SHOW command, 582
SHOW CREATE TABLE command, 584-585
SHOW DATABASES command, 583
SHOW GRANTS command, 583
SHOW INDEX command, 585
SHOW OPEN TABLES command, 584
SHOW STATUS command, 578-579, 587
SHOW TABLE STATUS command, 586
SHOW TABLES command, 584
SHOW VARIABLES command, 588
showcart.php script, 452-453
showitem.php script, 448-449
shuffle() function, arrays, 155
SHUTDOWN command, 38
signals, sending, 57
signatures, certificates, 565
signed data types, 306
signing requests, certificates, 571-572
SimpleXML functions, 535-537
single-byte character sets, 518

single-line comments, 81
sizeof() function, arrays, 154
slow queries status variable, 587

SMALLINT data type, 306
software
- configuring (Apache installations), 46-47
- load balancer (Apache performance), 550
- upgrades, 591-592

Solaris
- file descriptors, operating system scalability limits, 545
- server processes, operating systems (scalability), 544

source code
- Apache installation, 44
downloading (Apache installations), 45
uncompressing (Apache installations), 45-46

spaces (text)
- leading spaces, padding specifiers, 175
- multiple spaces, viewing in HTML, 175
- whitespace, deleting from strings, 185

special characters, padding specifiers, 175
specifying, virtual servers
  <VirtualHost> directive container, 52
split-file Perl script, 510
splitting logs, 510

SSH (Secure Shell), 34
SSL (Secure Sockets Layer)
  configuring (secure servers), 572
digital certificates, 565-566
installing
  mod ssl module, 568-569
  OpenSSL library, 567
OpenSSL installations
  Linux/UNIX, 568
  Windows, 567
protocols
  authentication, 564-567
  confidentiality, 562-563
  encryption, 562
SSLCertificateFile directive, 573
SSLeay libraries, 567
stacking images, 281
standard tags, 77
Start Apache link, 60
Start menu commands, 60
start session() function, 229
start tags, 77-78, 82
starting
  Apache, 58
    configuration file checks, 59
    manually, 49
  on Linux/UNIX, 60
  on Windows, 60
block of statements, 78
MySQL, 33-34
servers (SSL configurations), troubleshooting, 573
sessions, 228-229
state, saving, 208-209
static statements, remembering variable values between function calls, 141-142
status code, conditional logging, 505
storage (backend)
  database file-based access control authentication, 487
  file-based authentication, 485
  functions (authentication modules), 484
store categories field, storefront database table example, 434-435
store item color field, storefront database table example, 434-437
store item size field, storefront database table example, 434-436
store items field, storefront database table example, 434-436
stored procedures
  benefits of, 357
  syntax of, 358-359
storefront database table example
  add to cart button, 441-443
  cat id field, 434
  categories of items, displaying, 437-440
  planning process, 433
  store categories field, 434-435
  store item color field, 434-437
  store item size field, 434-436
  store items field, 434-436
storing
  certificate signing requests, 572
  formatted strings, 180
  password files (file-based authentication), 486
  session state, 228
  session variables, 229-231
str replace() function, 187
string data types, 91-92
string functions
  concatenation, 329-330
  length, 329-330
  location, 332
  modification, 334-335
  padding, 331-332
  position, 332
  substring, 333-334
  trimming, 331-332
string types
  BLOB, 308
  CHAR(M), 308
  ENUM, 309
  LONGBLOB, 309
  LONGTXT, 309
  MEDIUMBLOB, 309
  MEDIUXT, 309
  SET, 309
  TEXT, 308
  TINYBLOB, 309
  TINYTEXT, 309
  VARCHAR(M), 308
strings
  arrays, breaking into, 190-191
  cleaning up, 185
  defined, 79
  field names versus, 341
  formatting
    argument swapping, 179-180
    field width specifiers, 175-178
    printf() function, 172-175
    storing, 180
  HTML tags, removing from, 185
  indexing, 180-181
  length of, finding, 181
  log formats, 502
  new lines, removing from, 185
  portions of, extracting, 182-183
  query strings, passing session ID in, 233
  replacing portions of, 186
  substrings
    finding, 181-182
    position of, finding, 182
    replacing, 187
  tabs, removing from, 185
text
  converting case of, 187-188
  wrapping, 189-190
tokenizing, 183
web resources, 197
whitespace, removing from, 185
strip tags() function, 185
stripslashes() function, 423, 426, 430
stripos() function, 181
strstr() function, 181-182
strtok() function, 183
strtolower() function, 188
strtoUpper() function, 188
substr() function, 182
substring functions, 333-334
substring() function, 333

subject information (certificates), 565
subscribers tables, creating (mailing list subscription mechanisms), 376
subscription forms, creating (mailing list subscription mechanisms), 377-380, 383
subscription mechanism (mailing lists), 375
  include files, creating, 376-377
  subscriber tables, creating, 376
  subscription forms, creating, 377-380, 383
subscription project, subscribe and unsubscribe requests, 379-380

 TABLE

table cache parameter, 578-579
table type variable, 588
tables
  auth users, 493
  cache parameter, 578-579
  calendar example, 462-472
  creating
    asterisks (*), 313
    CREATE TABLE command, 309-310
    INSERT command, 310-312
    JOIN command, 320-321
    LIMIT command, 314-315
    primary/unique keys, 433
    SELECT command, 312-314, 317-319
    syntax of, 309
    WHERE clauses, 315-317
  custom logs, 511
  code snippet, 512-513
  sample reports, 513-515
  flat, 298-299
  FLUSH TABLES command, 582
  modifying records
    DELETE command, 326-328
    REPLACE command, 325-326
    UPDATE command, 323-325
  multiple tables, selecting via SELECT command, 317-319
  OPTIMIZE TABLE command, 579
  priv tables, 35
  relationships, 294
    many to many, 296-297
    one to many, 296
    one to one, 295

How can we make this index more useful? Email us at indexes@samspublishing.com
tables

shopping cart database table example
  adding items to cart, 450-451
  checkout actions, 456-457
  checkout forms, 456
  field lengths, 446
  field names, 445-446
  integrating with storefront, 447-449
  removing items from cart, 454-455
  viewing cart, 452-454
SHOW COLUMNS command, 585
SHOW CREATE TABLE command, 584-585
SHOW INDEX command, 585
SHOW OPEN TABLES command, 584
SHOW STATUS command, 587
SHOW TABLE STATUS command, 586
SHOW VARIABLES command, 588
storefront database table example
  add to cart button, 441-443
  cat id field, 434
  displaying categories of items, 437-440
  planning process, 433
  store categories field, 434-435
  store item color field, 434-437
  store item size field, 434-436
  store items field, 434-436
subqueries, 322
telephone tables, online address books, 391
type variable, 588

tabs, removing from strings, 185
tags
  AS2, 77
  end tags, 82
  Script, 77
  short, 77
  short open tag switch, 77
  standard, 77
  start tags, 82
  start/end, 78
  strip tags() function, 185
XML, 531
tagWrap() function, 145-146
tail command-line utility, monitoring error logs (UNIX), 511
tar command, 23, 45
tarballs, 45-46
telephone tables, online address books, 391
ternary (?) operators, 116
test() function, 138
testing
  data types, 91-96
dates, 196
files, 246-248
functions, checking for availability, 145-146
PHP installation, 74
text
  custom text, image creation, 287-288
editors, modifying httpd.conf files, 59
formatting documents as, 175
string text
  converting case, 187-188
  wrapping, 189-190
TEXT data type, 308
Thawte, CA (Certification Authorities), 572
third normal forms, 298, 301

time(data type, 308
time FORMAT() function, 343
time stamps
  converting to dates, 192-195
  creating, 195
  defining, 191
TIME TO SEC() function, 347
time() function, 191, 226
times/dates
  calendar, 459
event additions, 465-472
HTML, 460-462
library, creating, 473-479
table, creating, 462-465
user input, 460
current times, retrieving, 191
data types, 308
functions, 342-343
  arithmetic, 344-345
  conversion, 346-347
days, 336-338
file validation, 246
hours, 340-341
minutes, 340-341
months, 338-339
seconds, 340-341
special, 346-347
weeks, 339-340
years, 339
HH:MM:SS time format, 346
testing dates, 196
time stamps
  converting to dates, 192-195
  creating, 195
  defining, 191
UNIX, 191
web resources, 197
YYYY-MM-DD date format, 346

TimeOut directive, Apache performance, 551
TIMESTAMP data type, 308
TINYBLOB data type, 309
TINYINT data type, 306
TINYTEXT data type, 309
TITLE element, 461
TLS (Transport Layer Security), 562
tokenizing strings, 183
topic lists, displaying in discussion forums, 421-423
topic posts, displaying in discussion forums, 424-426
touch() function, 248
tracking clients
requests, access logs, 56
troubleshooting, 505
TRAILING function, 332
transactions, 353
  COMMIT command, 354-356
  examples of, 355-357
  ROLLBACK command, 354-356
  syntax of, 354
TransferLog directive, 505-506
transmitted data, reducing (Apache performance), 550
transparent images, 281
trim() function, cleaning up strings, 185
trimming functions, 331-332
Triple-Des, symmetric cryptography, 562
troubleshooting
  Apache startups, 61
  clients, tracking, 505
  installations, 20
  MySQL
    installation, 32
    upgrades, 593
  servers, starting (SSL configurations), 573
  TYPE argument, 218
type specifiers, printf() function, 173
Typical installation option (MySQL Setup Wizard), 26

U
UCASE() function, 334
ucfirst() function, 188
ucwords() function, 188
ulimit command, operating system scalability, 544
uncompressing source code (Apache installations), 45-46
underline() function, 145-146
UNIX
  Apache
    installation, 8-9, 45-47
    startups, 60
    apachectl tool, 58
    column command, passing data to, 261
    directories, creating listings for, 262
    epochs, 191
    FROM UNIXTIME() function, 347
    logresolve utility, resolving hostnames, 509
    mod_ssl Apache modules, 569
    MySQL, installing on, 7, 22-23
    OpenSSL libraries, installing, 568-569
  PHP
    Apache integration, 68-70
    installing, 9-10, 16, 66-68
    rotatelogs programs, 510
    syslog daemon, logging errors, 507-508
    tail command-line utility, monitoring error logs, 511
  UNIX TIMESTAMP() function, 347
  who command, reading output of, 260
  unlink() function, 248
  unsigned data types, 306
  unsubscribe requests, 379-380
  unsubscribe/subscribe forms, creating (mailing list subscription mechanisms), 379-380, 383
  unzipping software, 72
  UPDATE command, 38, 323-325, 365
    conditional UPDATE statements, 324
    subqueries, 322
  upgrades
    Apache, 593
    Apache News and Announcements list, 591
    modifying without upgrading, 594
    changelogs, 592
    determining when to upgrade, 592
    maintenance releases, 592
    MySQL, 593
    MySQL Announcements list, 591
    PHP, 595
    PHP Announcements list, 591
    security issues, 592
    staying current, 591
    when to upgrade, 592
  uploaded files, naming, 220
  uptime status variable, 587
  URL (Uniform Resource Locators), applying directives, 52
  User-Agent headers, 489
  user-defined functions, 133
  usernames, basic authentication, 482

How can we make this index more useful? Email us at indexes@samspublishing.com
users

adding, 37-39, 487
deleting, database file-based access control authentication, 487
input
calendar example, 459-460
creating images from, 282-285
input forms (HTML), 201-204
lists, Require directive, 483
login forms, 494-496
management
client authentication, 483
database file-based access control authentication, 487
file-based authentication, 485
functions, authentication modules, 484
ownership, verifying, 33
redirecting in HTML/PHP combination forms, 209-211
root users, running MySQL as, 40
sessions, 227
accessing variables, 229-232
adding arrays to, 231
destroying, 234
ID, passing in query strings, 233
registered users, 235
removing, 234
resuming, 228-229
session id() function, 228
session save path() function, 230
session set save handler() function, 228
start session() function, 229
starting, 228-229
storing, 229
storing state, 228
user preferences, 235
tables, 35
user files, backend storage, 485
USR1, signals, sending, 57
validating
directories, file/directory confirmation, 244
files
checking existence of, 244
date/time information, 246
determining file size, 245
file status, 245
file/directory confirmation, 244
testing functions, 246-248
values
directives, 55, 73
port values (Listen directive), 59
var keyword, object properties, 161
VARCHAR(M) data type, 308
variables
$count variable, 464
$dayArray variable, 464
$file array variable, 220
$fileName variable, 220
$firstDayArray variable, 465
$name variable, 474
$newnum variable, 133
$start variable, 464
$txt variable, 135
array-specific operations, 90
assignment operators ( =), 89
availability of, 89
casting, 94-95
data types, 90
array, 91
boolean, 91
changing, 93-95
double, 91-92
float, 91
integer, 91-92
NULL, 91
object, 91
resource, 91
string, 91-92
testing, 91-96
declaring, 89
outside of functions, 138
within functions, 137
defining, 87
DISTINCT, 339
environment variables
access control rules, 489
conditional looping, 506
globals, 89
accessing via global statements, 139
changing within functions, 140
HTTP_COOKIE, 225
integers, incrementing/decrementing, 100-101
local variables, 89
naming, 88
passing references to functions, 143-144
remembering values between function calls
   global statements, 140
   static statements, 141-142
scope of, 137
session variables
  accessing stored variables, 229-231
  removing, 234
  storing in, 229
superglobals, 89
$ COOKIE, 225
$ FILES, 217
$SESSION, 229-232, 235
values given to, overview, 88
when to use, 88
VeriSign, CA (Certification Authorities), 572
version upgrades, 592
version type variable, 588
virtual hosting, 552
virtual servers, specifying
  (<VirtualHost> directive container), 52
VirtualDocumentRoot directive (mass virtual hosting), 556
VirtualDocumentRootIP directive (mass virtual hosting), 556
VirtualScriptAlias directive (mass virtual hosting), 556
VirtualScriptAliasIP directive (mass virtual hosting), 556
virtual hosting
  DocumentRoot, 553
  IP-based, 553
  mass hosting, 555-556
  name-based, 553-555
VirtualHost containers, IP-based virtual hosting, 553
W
w (write) mode, 249
web pages, application localization, 521-524
web servers
  Apache, installing (Windows), 48
  logging and monitoring activity
    code snippet, 512
    CustomLog directive, 506
    database table creation, 511
    error logs, 507
    file accesses, 505-506
    HostNameLookup directive, 504
    identities, resolving, 509
    IdentityCheck directive, 505
    log analysis, 510-511
    log rotation, 509-510
    LogLevel directive, 508
    merging and splitting logs, 510
    program access, 506-507
    request logs, 501
    sample reports, 513-515
    status code, 505
    syslog daemon argument, 508
    what to log, 502-504
web spiders/crawlers, 551
Webalizer log analysis, 511
websites
  Apache, 45
  Apache Software Foundation, 44
  awstats, 511
  Logscan, 511
  MySQL, 22
  PHP 6.6, 71, 75
  PHP Manual, 371
  ScanErrLog, 511
  Thawte, 572
  VeriSign, 572
  Webalizer, 511
week functions, 339-340
WEEKDAY() function, 336
WHERE clauses, 315, 376
  comparison operators, 316
  logical operators, 316
  string comparisons, 317
while loops, 261, 426
while statements, 101, 117-118, 251
whitespace, 73, 185
who command (UNIX), reading output of, 260
wildcards, 38
WINCH signals, sending, 57
Windows
  Apache
    controlling (commands), 58
    installation, 13-14, 48-50
    PHP integration, 72-73
    startups, 60
  logresolve.exe utility, resolving hostnames, 509
  mod_ssl Apache modules, 569
  MySQL installations, 11, 26-27, 30-31
  OpenSSL libraries, installing, 567
  PHP
    Apache integration, 72-73
    installation, 15-16, 71-72
    rotatelogs.exe programs, rotating logs, 510
wizard
  MySQL Configuration Wizard, 27, 30-31
  MySQL Installation Wizard, 26
  MySQL Setup Wizard, 26

How can we make this index more useful? Email us at indexes@samspublishing.com
wordwrap() function

wordwrap() function, 189-190
wrapping string text, 189-190
write (w) mode, 249
writing files, 255

X

x-axis coordinates, drawing images, 272
X.509 digital certificates, 565
XML (Extensible Markup Language)
  case-sensitivity, 531
  children, 530
  content structure, 530
  document structure, 529-531
  Facebook, 531-532
  PHP accessing from
    DOM functions, 532-533
    SimpleXML functions, 535-537
  prologs, 529
  root elements, 530
  tags, 531
  uses of, 531-532
xor operators, 103

Y - Z

y-axis coordinates, drawing images, 272
year functions, 339
year select() function, 477-478
YEAR() function, 339
YEAR(M) data type, 308
YYYY-MM-DD date format, 346
zlib libraries, image creation, 271