

Christian Wenz



ESSENTIAL CODE AND COMMANDS

Updated
for
PHP 5.4

PHP and MySQL™

PHRASEBOOK



FREE SAMPLE CHAPTER



SHARE WITH OTHERS

PHP and MySQL™

P H R A S E B O O K

Christian Wenz

↕ Addison-Wesley

**Developer's
Library**

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

PHP and MySQL™ Phrasebook

Copyright © 2013 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-321-83463-8

ISBN-10: 0-321-83463-1

The Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

First Printing: October 2012

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearsoned.com

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Keith Cline

Indexer
Tim Wright

Proofreader
Kathy Ruiz

Technical Editor
John Coggeshall

**Publishing
Coordinator**
Vanessa Evans

Designer
Chuti Prasertsith

Page Layout
Studio Galou

Table of Contents

Introduction	1
1 Manipulating Strings	7
Comparing Strings	8
Checking Usernames and Passwords	9
Converting Strings into HTML	11
Using Line Breaks	13
Encrypting Strings	14
Checksumming Strings	16
Extracting Substrings	19
Protecting Email Addresses Using ASCII Codes	20
Scanning Formatted Strings	24
Getting Detailed Information about Variables	25
Searching in Strings	26
Using Perl-Compatible Regular Expressions	29
Finding Tags with Regular Expressions	30
Validating Mandatory Input	31
Validating Email Addresses	34
Search and Replace	35
2 Working with Arrays	39
Accessing All Elements of Numeric Arrays	41
Accessing All Elements of Associative Arrays	43
Accessing All Array Elements in Nested Arrays	44
Turning an Array into Variables	47
Converting Strings to Arrays	48
Converting Arrays to Strings	49
Sorting Arrays Alphabetically	50
Sorting Associative Arrays Alphabetically	51
Sorting Nested Arrays	53
Sorting Nested Associative Arrays	55

Sorting IP Addresses (as a Human Would)	57
Sorting Anything	58
Sorting with Foreign Languages	59
Applying an Effect to All Array Elements	61
Filtering Arrays	65
Getting Random Elements Out of Arrays	67
Making Objects Behave Like Arrays	68
3 Date and Time	71
Using Text within date()	74
Formatting DateTime Objects	75
Automatically Localizing Dates	75
Manually Localizing Dates	79
Using the Current Date, the U.S./U.K./European Way	80
Formatting a Specific Date	80
Validating a Date	82
Calculating a Relative Date	83
Creating a Sortable Time Stamp	84
Converting a String into a Date	85
Determining Sunrise and Sunset	86
Using Date and Time for Benchmarks	87
Using Form Fields for Date Selection	89
Create Self-Updating Form Fields for Date Selection	91
Calculating the Difference between Two Dates	93
Using GMT Date/Time Information	96
4 Working with Objects (and Related Topics)	99
Setting Up Classes	100
Understanding Inheritance	102
Using Abstract Classes and Interfaces	104

Preventing Inheritance and Overwriting	108
Using Autoloading	109
Cloning Objects	111
Serializing and Deserializing Objects	113
Implementing Singletons	115
Using Traits	120
5 Interacting with Web Forms	123
Sending Form Data Back to the Current Script	124
Reading Out Form Data	125
Checking Whether a Form Has Been Submitted	128
Saving Form Data into a Cookie	129
Prefilling Text Fields and Password Fields	131
Prefilling Multiline Text Fields	135
Preselecting Radio Buttons	136
Preselecting Check Boxes	137
Preselecting Selection Lists	138
Preselecting Multiple Selection Lists	140
Processing Graphical Submit Buttons	143
Checking Mandatory Fields	144
Checking Selection Lists	146
Escaping Output	149
Validating Input	150
Writing All Form Data into a File	151
Sending All Form Data via Email	153
Getting Information about File Uploads	154
Moving Uploaded Files to a Safe Location	157
Monitoring the Progress of a File Upload	158
6 Remembering Users (Cookies and Sessions)	163
Understanding Cookies	164

Creating a Cookie	167
Reading Out Cookies	169
Setting a (Reasonable) Expiry Date	170
Setting a Client-Specific Expiry Date	171
Deleting a Cookie	172
Making Cookies Accessible for Several Domains	174
Checking Whether the Client Supports Cookies	176
Saving Multiple Data in One Cookie	177
Saving the User's Language Preference	180
Understanding Sessions	182
Where to Store the Sessions	183
How to Maintain the Session State	184
Activating Sessions	185
Reading and Writing Sessions	186
Closing Sessions	187
Changing the Session ID	187
Implementing a Custom Session Management	189
Creating a Secured Area with Sessions	195
Creating a Secured Area without Sessions	197
7 Using Files on the Server File System	201
Opening and Closing Files	202
Reading from Files	205
Writing to Files	207
Locking Files	208
Using Relative Paths for File Access	209
Avoiding Security Traps with File Access	210
Working with CSV Data	212
Parsing INI Files	216
Retrieving File Information	217

Copying, Moving, and Deleting Files	220
Browsing the File System	221
Using PHP Streams	222
Using Bzip2 Archives	224
Returning Files with an HTTP Request	227
8 Working with MySQL Databases	229
Connecting to MySQLi	231
Sending SQL to MySQL	233
Prepared Statements with MySQL	235
Retrieving Results of a Query to MySQL	236
Retrieving the Last Inserted ID	239
Using Transactions	240
9 Working with Other Databases	243
Connecting to SQLite	244
Sending SQL to SQLite	246
Retrieving Results of a Query to SQLite	248
Using Prepared Statements with SQLite	250
Connecting to PostgreSQL	252
Sending SQL to PostgreSQL	253
Updating Data in PostgreSQL	254
Retrieving Results of a Query to PostgreSQL	255
Connecting to Oracle	257
Sending SQL to Oracle	258
Retrieving Results of a Query to Oracle	260
Connecting to MSSQL	262
Sending SQL to MSSQL	264
Retrieving Results of a Query to MSSQL	265
Using Prepared Statements with MSSQL	266
Using MSSQL without Windows	268
Connecting to Firebird	270
Sending SQL to Firebird	271

Retrieving Results of a Query to Firebird	272
Connecting via PDO	273
Sending SQL via PDO	275
Retrieving Results of a Query via PDO	276
10 Using XML	279
Parsing XML with SAX	280
Parsing XML with XMLReader	282
Using DOM to Read XML	284
Using DOM to Write XML	285
Using XMLWriter to Write XML	287
Using SimpleXML	288
Using XPath with SimpleXML	289
Transforming XML with XSL	290
Validating XML	291
11 Communicating with Others	295
Connecting with HTTP Servers	295
Connecting with FTP Servers	298
Checking Whether a Server Is Still Reacting	300
Creating a Web Service with NuSOAP	304
Automatically Generating WSDL with NuSOAP	306
Consuming a Web Service with NuSOAP	308
Creating a Web Service with the PHP 5 SOAP Extension	309
Consuming a Web Service with the PHP 5 SOAP Extension	312
Using Ajax	313
Exchanging Data with the Server	316
Index	321

About the Author

Christian Wenz is a professional phrasemonger, author, trainer, and consultant with a focus on web technologies. He has written or cowritten more than one hundred books. He frequently contributes articles to renowned IT magazines and speaks at developer conferences around the globe. Christian contributes to several PHP libraries and frameworks and other open source software. He holds university degrees in computer sciences and in business informatics and lives and works in Munich, Germany. He also is one of the authors of Zend's PHP 5 and PHP 5.3 certifications.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and contact information.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our Web site and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Some time ago, my favorite development editor, Damon Jordan, sent me an email and closed it with “*Ich möchte eine Föhnwelle*,” which means “I’d like a blow wave.” Unfortunately, I didn’t know what either a *Föhnwelle* or a blow wave was, so I declined. He then told me he had found this sentence in a German phrasebook he recently bought.

I was interested and had a look at some German phrasebooks. I think they are great tools to get around in a foreign country, although I personally think that some of the phrases offered just don’t make sense. For instance, in one phrasebook, I found a series of pickup lines, including the ingenious “You have a beautiful personality,” something that didn’t work for me either in English, in German, or in any other language! Some coital guidance could also result in other problems—you either have to remember all the things to say while you are at it, or you have to hold the phrasebook in your free hand. And, finally, “Blow waves are as dead as a pet rock,” just to use another phrase.

Anyway, we were discussing phrasebooks a bit, and Damon said that he wanted to do a book series on phrasebooks. He also mentioned that he would like to team up with his and my favorite acquisitions editor, Shelley Johnston, so I was in.

While working on a concept, we found some differences between a language phrasebook and an IT

phrasebook. For instance, a language phrasebook just contrasts the same sentence in two languages.

However, this is not always helpful. What if you want to change the phrase a bit, for instance if you want an en vogue blow wave (an oxymoron, one might say)?

So, we tried to create a concept that contains a lot of phrases, but all of them with good explanations so that it is easy to change the code and adapt it to one's needs. This, of course, makes the "foreign language" portions of a phrase a bit longer than the phrase itself, but we think that really helps when working with the book.

I also remember one famous Monty Python sketch in which someone uses a sabotaged dictionary, so that asking for directions results in getting roughed up. Therefore, it is vitally important to get a real explanation on what is going on within the phrase.

I then wrote a series concept and a sample chapter, and now, only a few months later, you hold the first phrasebook in your hands (one of many, we hope).

Something I really hate about reading computer books is when code samples are hacked into the word processor but never tested. To avoid this, every listing is also available for download at

<http://php.phrasebook.org/>; the filename is part of the listing's caption. So every code sample does exist as a file and has actually been tested, unlike in some other books. Of course, it's an illusion that this book is 100 percent error free, although we have taken several steps to come very close to that mark. Any errata, if known, will be posted to that site, too.

Another thing I really dislike with some books is that they tend to be very OS dependent, which is really

unnecessary for PHP. Some books were obviously only tested under Windows, some others only under Linux, but it is possible to make code relatively platform independent. We have invested a lot of effort in testing the code from this book on many server platforms, including Linux, Windows, Mac OS X, and Solaris. Therefore, the screenshots in this book are also taken from those platforms, so you will find a healthy mixture of systems (and browsers). Ideology can be expressed with many phrases, but you won't find any of them in this book. If something does run only on certain platforms (or PHP versions), it is noted in the text. Another phrase I promise you will not find in this book is anything that looks like `foo`, `bar`, `baz`, or any other proofs of very little imagination.

Of course, it is easy to find missing phrases in this book. PHP offers so much functionality that it is impossible to cover every aspect. Therefore, we had to select certain topics of interest—stuff that is relevant in a PHP programmer's everyday work. If you think, however, that something has really been overlooked, please let me know (but do also nominate something that should then be removed from upcoming editions of this book to make room for the new phrases). I am looking forward to getting your feedback.

And now, to quote a phrasebook once more: “*Bist du soweit? Da boxt der Papst.*” That is, “Are you ready? It's all happening there” (but literally, “There boxes the pope.”)

Your personal phrasemonger,

Christian Wenz

Introduction for the Second Edition

When we worked on the concept of the Phrasebook series a few years ago, we were quite confident that the books planned for the first batch would do well.

Looking back now, with a dozen books still out there, and quite a number of foreign-language translations, I can say we did *really* well. It also makes me personally quite proud that the pilot, the *PHP Phrasebook*, still stands the test of time, and I am happy to see this book quite often when I am visiting a customer's site.

However, PHP has progressed quite a bit in the past few years. PHP 4 has finally vanished, and although the promised PHP 6 never materialized, PHP versions 5.3 and 5.4 added many new features, a lot of them originally planned for the next major PHP update.

We took this opportunity to update the *PHP Phrasebook* to the latest developments in PHP. Because we got really stellar feedback for the original book, we tried to leave the setup and popular phrases intact, but we added dozens of new and updated phrases, highlighting many of the new possibilities of PHP 5.3 and 5.4. For the sake of backward compatibility, we always mention in which version of PHP a new feature has

been introduced, and if possible, we provide code that does not necessarily require the latest and greatest, if we see it fit.

The name of the book changed a bit, as well. We covered many relevant databases in the first edition, and still do now; however, MySQL continues to be the de facto standard database for PHP development, and therefore it gets its own chapter and part of the book's title page.

I am indebted to the many readers who provided me with suggestions, errata, and general feedback. Of course, any kind of feedback for this edition of the book is greatly appreciated, as well. The Web site at <http://php.phrasebook.org/> contains the code samples from this book, errata (as soon as I learn about them), and contact information. I am looking forward to hearing from you!

As with most books, this has been a team effort. Mark Taber, who oversaw the creation of the Phrasebook series back then, is still onboard and was the project manager for this edition. I still remember when we drafted the new edition during breakfast on one of the rare occasions when we actually met in person. (Next time I'll pay.) I was happy that my old friend John Coggeshall was kind enough to serve as a technical reviewer—and to save my reputation a few times. Also “thank you” to the team of wizards at Pearson who turn my manuscripts into professional books. And thanks to my family and friends who accept the odd hours my profession requires from time to time.

This page intentionally left blank

Working with Arrays

When simple variables are just not good enough, arrays come into play (or objects, but that's another topic). The array section in the PHP manual, available at <http://php.net/array>, lists approximately 80 functions that are helpful. Therefore, this book could be filled with array-related phrases alone. However, not all of these functions are really used often. Therefore, this chapter presents the most important problems you'll have to solve when working with arrays—and, of course, solutions for these problems.

There are two types of arrays. The names they are given differ sometimes, but usually arrays are distinguished between numeric arrays and associative arrays. The first type of array uses numeric keys, whereas the latter type can also use strings as keys.

Creating an array can be done in one of three ways:

- Using the `array()` statement

```
$a = array('I', 'II', 'III', 'IV');
```

- Successively adding values to an array using the variable name and square brackets

```
$a[] = 'I';
$a[] = 'II';
$a[] = 'III';
$a[] = 'IV';
```

- Using a new square brackets syntax introduced in PHP 5.4

```
$a = ['I', 'II', 'III', 'IV'];
```

The latter method is probably the most intuitive one for Web developers because JavaScript features a very similar syntax. For the sake of backward compatibility, we will still use the first option throughout this chapter.

When using associative arrays, the same three methods can be used; however, this time keys and values must be provided:

```
$a1 = array('one' => 'I', 'two' => 'II', 'three' =>
↳ 'III', 'four' => 'IV');
$a2['one'] = 'I';
$a2['two'] = 'II';
$a2['three'] = 'III';
$a2['four'] = 'IV';
```

```
$a1 = ['one' => 'I', 'two' => 'II', 'three' =>
↳ 'III', 'four' => 'IV'];
```

Arrays can also be nested, when an array element itself is an array:

```
$a = array(
    'Roman' =>
        array('one' => 'I', 'two' => 'II', 'three' =>
            'III', 'four' => 'IV'),
    'Arabic' =>
        array('one' => '1', 'two' => '2', 'three' =>
            '3', 'four' => '4')
);
```

Now, the Arabic representation of the number four can be accessed using `$a['Arabic']['four']`.

Of course, arrays are not only created within a script but can also come from other sources, including from HTML forms (see Chapter 5, “Interacting with Web Forms”) and from cookies and sessions (see Chapter 6, “Remembering Users (Cookies and Sessions)”). But if the array is there, what’s next? The following phrases give some pointers.

Accessing All Elements of Numeric Arrays

```
foreach ($a as $element)
```

```
<?php
    $a = array('I', 'II', 'III', 'IV');
    foreach ($a as $element) {
        echo htmlspecialchars($element) . '<br />';
    }
?>
```

Looping through an Array with foreach (foreach-n.php)

Looping through numeric (or indexed) arrays can most easily be done using `foreach` because in each

iteration of the loop, the current element in the array is automatically written in a variable, as shown in the preceding code.

Alternatively, a `for` loop can also be used. The first array element has the index 0; the number of array indices can be retrieved using the `count()` function:

```
<?php
    $a = array('I', 'II', 'III', 'IV');
    for ($i = 0; $i < count($a); $i++) {
        echo htmlspecialchars($a[$i]) . '<br />';
    }
?>
```

Looping through an Array with for (for-n.php)

Both ways are equally good (or bad); though, usually, using `foreach` is the much more convenient way. However, there is a third possibility: The PHP function `each()` returns the current element in an array. The return value of `each()` is an array, in which you can access the value using the numeric index 1, or the string index 'value'. Using a `while` loop, the whole array can be traversed. The following code once again prints all elements in the array, this time using `each()`:

```
<?php
    $a = array('I', 'II', 'III', 'IV');
    while ($element = each($a)) {
        echo htmlspecialchars($element['value']) .
        ↪'<br />'; //or: $element[1]
    }
?>
```

Looping through an Array with each (each-n.php)

The output of the three listings is always the same, of course.

Accessing All Elements of Associative Arrays

```
foreach ($a as $key => $value)
```

```
<?php
    $a = array('one' => 'I', 'two' => 'II', 'three' =>
    └─'III', 'four' => 'IV');
    foreach ($a as $key => $value) {
        echo htmlspecialchars("$key: $value") . '<br
    └─/>';
    }
?>
```

*Looping through an Associative Array with foreach
(foreach-a.php)*

When using an associative array and wanting to access all data in it, the keys are also of relevance. For this, the `foreach` loop can also provide a variable name for the element's key, not only for its value.

Using `count()` is possible: `count()` returns the number of values in the array, not the number of elements. Looping through all array elements with `for` is not feasible. However, the combination of `each()` and `while` can be used, as shown in the following code. The important point is that the key name can be retrieved either using the index 0 or the string index 'key':

```

<?php
    $a = array('one' => 'I', 'two' => 'II', 'three' =>
    ↪ 'III', 'four' => 'IV');
    while ($element = each($a)) {
        echo htmlspecialchars($element['key'] . ': ' .
    ↪ $element['value']) . '<br />';
        //or: $element[0] / $element[1]
    }
?>

```

Looping through an Associative Array with each (each-a.php)

Accessing All Array Elements in Nested Arrays

```
print_r($a);
```

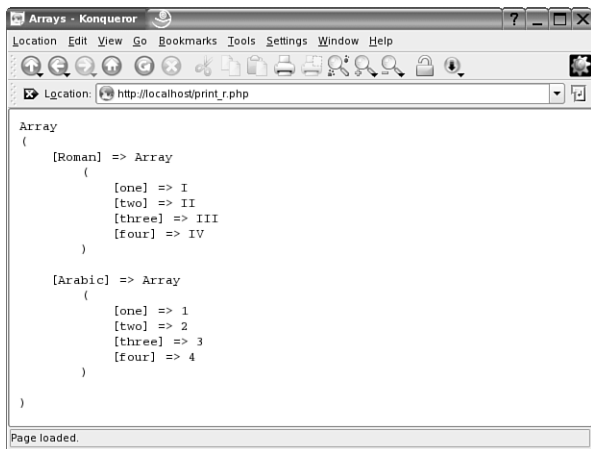
```

<pre>
<?php
    $a = array(
        'Roman' =>
            array('one' => 'I', 'two' => 'II', 'three' =>
    ↪ 'III', 'four' => 'IV'),
        'Arabic' =>
            array('one' => '1', 'two' => '2', 'three' =>
    ↪ '3', 'four' => '4')
    );
    print_r($a);
?>
</pre>

```

Printing a Nested Array with print_r (print_r.php)

Nested arrays can be printed really easily by using `print_r()`. Take a look at the output of the listing in Figure 2.1.



```
Array
(
    [Roman] => Array
        (
            [one] => I
            [two] => II
            [three] => III
            [four] => IV
        )

    [Arabic] => Array
        (
            [one] => 1
            [two] => 2
            [three] => 3
            [four] => 4
        )
)
```

Page loaded.

Figure 2.1 Printing array contents with `print_r()`

TIP: If you set the second parameter of `print_r()` to true, the associative array's contents are not sent to the client, but are returned from the function, so that you can save this information in a variable and process it further.

However, the output of the preceding code (see Figure 2.1) is hardly usable for more than debugging purposes. Therefore, a clever way to access all data must be found. A recursive function is a reasonable way to achieve this. In this, all elements of an array are printed out; the whole output is indented using the HTML element `<blockquote>`. If the array element's value is an array itself, however, the function calls itself recursively, which leads to an additional level of indentation.

Whether something is an array can be determined using the PHP function `is_array()`. Using this, the following code can be assembled; see Figure 2.2 for the result:

```
<?php
function printNestedArray($a) {
    echo '<blockquote>';
    foreach ($a as $key => $value) {
        echo htmlspecialchars("$key: ");
        if (is_array($value)) {
            printNestedArray($value);
        } else {
            echo htmlspecialchars($value) . '<br />';
        }
    }
    echo '</blockquote>';
}

$arr = array(
    'Roman' =>
        array('one' => 'I', 'two' => 'II', 'three' =>
    ➤ 'III', 'four' => 'IV'),
    'Arabic' =>
        array('one' => '1', 'two' => '2', 'three' =>
    ➤ '3', 'four' => '4')
);

printNestedArray($arr);
?>
```

*Printing a Nested Array Using a Recursive Function
(printNestedArray.php)*

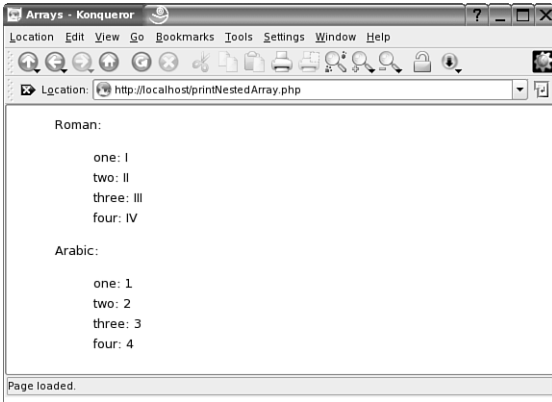


Figure 2.2 Printing array contents using a recursive function

Turning an Array into Variables

```
while (list($key, $value) = each($a))
```

```
<?php
    $a = array('one' => 'I', 'two' => 'II', 'three' =>
    'III', 'four' => 'IV');
    while (list($key, $value) = each($a)) {
        echo htmlspecialchars("$key: $value") . '<br
    />';
    }
?>
```

*Looping through an Array with list() and each()
(each-list.php)*

Whenever `each()` is used, the use of `list()` is a good idea. Within `list()`, you provide variable names for all values with numeric indices in the array that is returned by `each()`. This makes `while/each()` loops even easier to use, as the code shows. Within the parentheses, the variable names are provided.

TIP: If you are interested only in either the key or the value of the array, you can remove one of the two variables; just make sure that you keep the comma.

```
while (list(, $value) = each($a)) {
    echo htmlspecialchars("$value") . '<br />';
}
```

Converting Strings to Arrays

```
$a = explode(',', $csvdata);
```

```
<?php
    $csvdata = 'Pearson Education,800 East 96th
    Street,Indianapolis,Indiana,46240';
    $a = explode(',', $csvdata);
    $info = print_r($a, true);
    echo "<pre>$info</pre>";
?>
```

Turning a String into an Array (explode.php)

Sometimes, arrays are not used to store information; instead, a string is used. The single values are all within the string, but are separated by a special character. One example for this is the comma-separated values (CSV) format.

The PHP function `explode()` creates an array out of these values; you just have to provide the characters at which the string needs to be split. The browser then shows this output:

```
Array
(
    [0] => Pearson Education
    [1] => 800 East 96th Street
    [2] => Indianapolis
    [3] => Indiana
    [4] => 46240
)
```

Converting Arrays to Strings

```
$address = implode('<br />', $data);
```

```
<?php
    $data = array(
        'Pearson Education',
        '800 East 96th Street',
        'Indianapolis',
        'Indiana',
        '46240'
    );
    $address = implode('<br />', $data);
    echo $address;
?>
```

Turning an Array into a String (implode.php)

The way back (that is, making a string representation out of the elements in an array) can be done using `implode()`. Again, two parameters are required: the separation elements, then the array. The order is quite unusual, yet important.

So, PHP joins the elements of the array, using the `
` HTML element. Therefore, in the browser, each array's elements stay at its own line.

Sorting Arrays Alphabetically

```
sort($a, SORT_NUMERIC);  
sort($a, SORT_STRING);
```

```
<pre>  
<?php  
    $a = array('4', 31, '222', 1345);  
    sort($a, SORT_NUMERIC);  
    print_r($a);  
    sort($a, SORT_STRING);  
    print_r($a);  
?>  
</pre>
```

Sorting an Array (sort.php)

Numeric arrays can be sorted rather easily by using `sort()`. However, a problem exists if the array contains both numeric and string values (for instance, "2" > "10" but $2 < 10$). Therefore, the sorting can be tweaked so that a special data type is used for comparing elements when sorting:

- `SORT_NUMERIC` sorts elements as numbers.
- `SORT_REGULAR` sorts elements according to their data type (standard behavior).
- `SORT_STRING` sorts elements as strings.

Here is the output of the preceding listing:

```
Array  
(
```

```

    [0] => 4
    [1] => 31
    [2] => 222
    [3] => 1345
)
Array
(
    [0] => 1345
    [1] => 222
    [2] => 31
    [3] => 4
)

```

NOTE: If you want to sort the elements of the array in reverse order, use `rsort()` (*r* for reverse). The same optional second parameters are allowed that can be used with `sort()`.

Sorting Associative Arrays Alphabetically

```

ksort($a);
asort($a);

```

```

<pre>
<?php
    $a = array('one' => 'I', 'two' => 'II', 'three' =>
    'III', 'four' => 'IV');
    ksort($a);
    print_r($a);
    asort($a);
    print_r($a);
?>
</pre>

```

Sorting an Associative Array (*sort_a.php*)

Sorting associative arrays can be done in one of several ways:

- Sort by keys, leave key-value association intact: Use `ksort()`.
- Sort by keys in reverse order, leave key-value association intact: Use `krsort()`.
- Sort by values, leave key-value association intact: Use `asort()`.
- Sort by values in reverse order, leave key-value association intact: Use `arsort()`.

The preceding code shows these functions in action; Figure 2.3 shows the result.

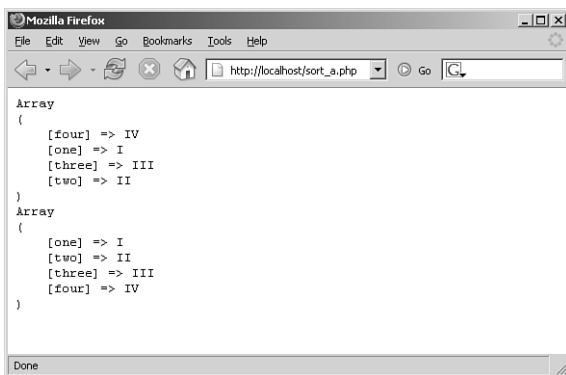


Figure 2.3 Sorting associative arrays

NOTE: Trying to use `sort()` or `rsort()` with associative arrays works, but the keys are then all lost.

Sorting Nested Arrays

```
function sortNestedArray(&$a) {
    sort($a);
    for ($i = 0; $i < count($a); $i++) {
        if (is_array($a[$i])) {
            sortNestedArray($a[$i]);
        }
    }
}
```

```
<pre>
<?php
    function sortNestedArray(&$a) {
        sort($a);
        for ($i = 0; $i < count($a); $i++) {
            if (is_array($a[$i])) {
                sortNestedArray($a[$i]);
            }
        }
    }

    $arr = array(
        'French',
        'Spanish',
        array('British English', 'American English'),
        'Portuguese',
        array('Schwitzerdütsch', 'Deutsch'),
        'Italian'
    );
    sortNestedArray($arr);
    print_r($arr);
?>
</pre>
```

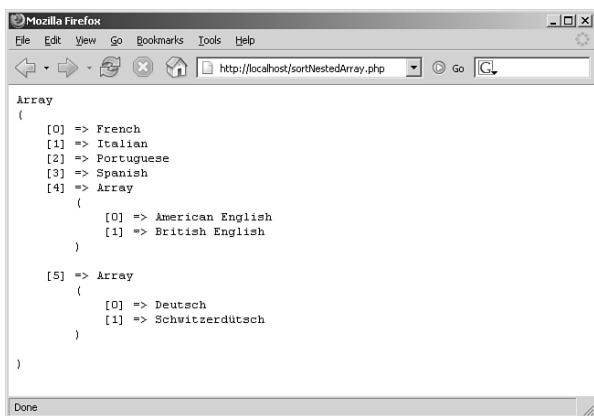
Sorting a Nested Array Using a Recursive Function
([sortNestedArray.php](#))

The standard sorting functions of PHP do not traverse nested arrays when performing their operations.

However, if you use a recursive function, you can code this in just a few lines.

The goal is to sort an array that is nested but consists only of numeric subarrays so that only numeric (and, therefore, useless) keys are used.

The idea is the following: Calling `sort()` does sort the array, but leaves out all subarrays. Therefore, for all elements that are arrays, the sorting function is called again, recursively. The preceding code shows this concept; Figure 2.4 shows the result for a sample array.



```
Array
(
    [0] => French
    [1] => Italian
    [2] => Portuguese
    [3] => Spanish
    [4] => Array
        (
            [0] => American English
            [1] => British English
        )
    [5] => Array
        (
            [0] => Deutsch
            [1] => Schwitzerdütsch
        )
)
```

Figure 2.4 Sorting nested arrays

NOTE: The PHP function `array_multisort()` is an alternative way to sort arrays with more than one dimension.

Sorting Nested Associative Arrays

```
foreach ($a as $key => $value) {
    if (is_array($value)) {
        sortNestedArrayAssoc($value);
    }
}
```

```
<pre>
<?php
function sortNestedArrayAssoc($a) {
    ksort($a);
    foreach ($a as $key => $value) {
        if (is_array($value)) {
            sortNestedArrayAssoc($value);
        }
    }
}

$arr = array(
    'Roman' =>
        array('one' => 'I', 'two' => 'II', 'three' =>
    'III', 'four' => 'IV'),
    'Arabic' =>
        array('one' => '1', 'two' => '2', 'three' =>
    '3', 'four' => '4')
);
sortNestedArrayAssoc(&$arr);
print_r($arr);
?>
</pre>
```

Sorting an Associative Nested Array Using a Recursive Function (sortNestedArrayAssoc.php)

If an associative nested array is to be sorted, two things have to be changed in comparison to the previous

phrase that sorted a numeric (but nested) array. First, the array has to be sorted using `ksort()`, not `sort()`. Furthermore, the recursive sorting has to be applied to the right variable, the array element that itself is an array. Make sure that this is passed via reference so that the changes are applied back to the value:

```
foreach ($a as $key => &$value) {
    if (is_array($value)) {
        sortNestedArrayAssoc($value);
    }
}
```

Figure 2.5 shows the result of the code at the beginning of this phrase.

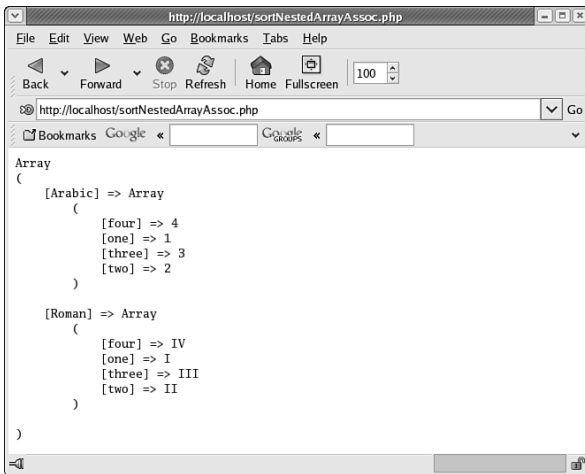


Figure 2.5 Sorting nested, associative arrays

Sorting IP Addresses (as a Human Would)

```
natsort($a);
```

```
<?php
    $a = array('100.200.300.400', '100.50.60.70',
    ─ '100.8.9.0');
    natsort($a);
    echo implode(' < ', $a);
?>
```

Sorting IP Addresses Using a Natural String Order String Comparison (natsort.php)

Sorting IP addresses with `sort()` does not really work because if sorting as strings, '100.200.300.400' (which intentionally is an invalid IP) is less than '50.60.70.80'. In addition, there are more than just digits within the string, so a numeric sorting does not work.

What is needed in this case is a so-called natural sorting, something that has been implemented by Martin Pool's Natural Order String Comparison project at <http://sourcefrog.net/projects/natsort/>. In PHP's `natscasesort()` function, this algorithm is used. According to the description, it sorts "as a human would." When case sensitivity is an issue, `natsort()` can be used. The preceding code shows the latter function.

NOTE: Internally, `natsort()` uses `strnatcmp()` (and `natscasesort()` uses `strnatcasecmp()`), which does a "natural" comparison of two strings. By calling this function a number of times, the array elements are brought into the correct order.

Sorting Anything

```
<?php
function compare($a, $b) {
    return $a - $b;
}

$a = array(4, 1345, 31, 222);
usort($a, 'compare');
echo implode(' < ', $a);
?>
```

If you do not want to limit yourself to the standard sorting functionality offered by PHP, you can write your own sorting algorithm. Internally, PHP uses the Quicksort algorithm to sort values in an array. For this to work, PHP has to know whether two values are equal; in the latter case, PHP needs to find out which value is greater. So, to implement a custom sort, all that is required is a function that takes two parameters and returns:

- A negative value if the first parameter is smaller than the second parameter
- 0 if both parameters are equal
- A positive value if the second parameter is smaller than the first parameter

The name of this function must be passed to `usort()`—as a string—or, if you are using at least PHP 5.3, you can also rely on an anonymous function similar to JavaScript. The rest of the work is done by PHP, as shown in the code. The comparison function used there is a very simple way to do a numeric sorting. By subtracting the two values, the function returns the desired values: A positive number if the first parameter is larger than the second one, 0 if both parameters are equal, and a negative number otherwise.

Sorting with Foreign Languages

```
<?php
function compare($a, $b) {
    if ($a == $b) {
        return 0;
    } else {
        for ($i = 0; $i < min(strlen($a), strlen($b));
        ↪$i++) {
            $cmp = compareChar(substr($a, $i, 1),
        ↪substr($b, $i, 1));
            if ($cmp != 0) {
                return $cmp;
            }
        }
        return (strlen($a) > strlen($b)) ? 1 : 0;
    }
}

function compareChar($a, $b) {
    // ...
}

$a = array('Frédéric', 'Froni', 'Frans');
usort($a, 'compare');
echo implode(' < ', $a);
?>
```

*Sorting an Array with Language-Specific Characters
(languagesort.php; excerpt)*

Sorting works well, as long as only the standard ASCII characters are involved. However, as soon as special language characters come into play, the sorting yields an undesirable effect. For instance, calling `sort()` on an array with the values 'Frans', 'Frédéric', and 'Froni' puts 'Frédéric' last because the é character has a much larger charcode than o.

For this special case, PHP offers no special sorting method; however, you can use `strnatcmp()` to emulate this behavior. The idea is to define a new order for some special characters; in the comparison function, you then use this to find out which character is “larger” and which is “smaller.”

You first need a function that can sort single characters:

```
function compareChar($a, $b) {
    $characters =
'AAÁÀBCÇDEÉÉFGHIÍĲJKLMNOÒÓPQRSTUÚÚVWXYZ';
    $characters .= 'aäääbcçdeëéfgghiïijklm
↳noòóöppqrstuúúvwxyz';
    $pos_a = strpos($characters, $a);
    $pos_b = strpos($characters, $b);
    if ($pos_a === false) {
        if ($pos_b === false) {
            return 0;
        } else {
            return 1;
        }
    } elseif ($pos_b === false) {
        return -1;
    } else {
        return $pos_a - $pos_b;
    }
}
```

Then, the main sorting function calls `compareChar()`, character for character, until a difference is found. If no difference is found, the longer string is considered to be the “greater” one. If both strings are identical, 0 is returned. The code at the beginning of this phrase shows the compare function. The result of this code is, as desired, Frans < Frédéric < Froni.

Starting with PHP 5.3, the `ext/intl` extension provides a mechanism for natural language sorting, as well. If the

extension is installed (which may additionally require the ICU library set from <http://site.icu-project.org/>), you first need to create a so-called collator, which expects a locale (for instance, en_US, en_CA, en_GB, fr_FR, or de_AT). Then, you can call the `sort()` and `asort()` methods, which work analogously to their PHP counterparts but take the locale information into account:

```
$a = array('Frédéric', 'Froni', 'Frans');
$coll = new Collator('fr_FR');
$coll->sort($a);
echo implode(' < ', $a);
```

Sorting an Array with Locale Information (collator.php)

Applying an Effect to All Array Elements

```
$a = array_map('sanitize', $a);
```

```
<?php
function sanitize($s) {
    return htmlspecialchars($s);
}

$a = array('harmless', '<bad>', '>>click
➔here!<<');
$a = array_map('sanitize', $a);
echo implode(' ', $a);
?>
```

Applying htmlspecialchars() to All Elements of an Array (array_map.php)

Sometimes, data in an array has to be preprocessed before it can be used. In Chapter 4, you will see how data coming from the user via HTML forms can be sanitized before it is used. To do so, every array element must be touched.

However, it is not required that you do a cumbersome `for/foreach/while` loop; PHP offers built-in functionality for this. The first possibility is to use `array_map()`. This takes an array (second parameter) and submits every element in that array to a callback function (first parameter, as a string, an array, or an anonymous function, as you can see below). At the end, the array is returned, with all of the elements replaced by the associated return values of the callback function.

In the preceding listing, all values in the array are converted into HTML using `htmlspecialchars()`.

NOTE: Starting with PHP 5.3, you may use inline anonymous functions whenever a callback is expected by a PHP function. So, the previous code could be cleaned up a bit like this:

```
<?php
    $a = array('harmless', '<bad>', '>>click
➤here!<<');
    $a = array_map(
        function sanitize($s) {
            return htmlspecialchars($s);
        },
        $a);
    echo implode(' ', $a);
?>
```

Applying `htmlspecialchars()` to All Elements of an Array, Using an Anonymous Function (`array_map_anon.php`)

If the array turns out to be a nested one, however, the tactic has to be changed a little. Then you can use a recursive function. If an array element is a string, it is HTML encoded. If it's an array, the recursive function calls itself on that array. The following code implements this, and Figure 2.6 shows the result:

```
<?php
function sanitize_recursive($s) {
    if (is_array($s)) {
        return(array_map('sanitize_recursive', $s));
    } else {
        return htmlspecialchars($s);
    }
}

$a = array(
    'harmless' =>
        array('no', 'problem'),
    'harmful' =>
        array('<bad>', '-> <worse> <<-')
);

$a = sanitize_recursive($a);
echo '<pre>' . print_r($a, true) . '</pre>';
?>
```

Recursively Applying htmlspecialchars() to All Elements of a Nested Array (array_map_recursive.php)

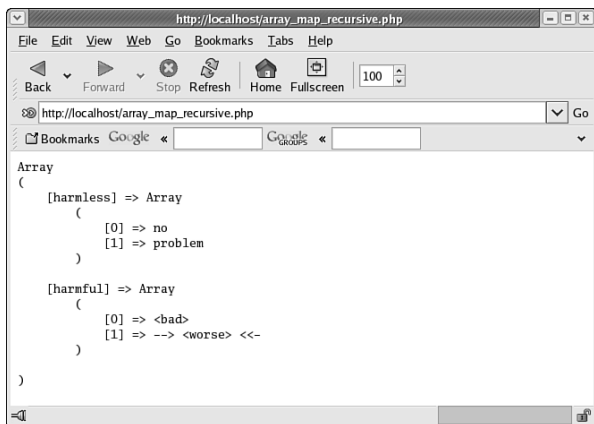


Figure 2.6 The nested arrays have been HTML-encoded.

Another function that behaves similarly is `array_walk()`. This one also applies a function to every element of an array; however, it also allows you to provide a parameter for this function call. In the following code, this is used to print out all elements of an array. A parameter is passed—a counter. Because it is passed by reference, increasing this counter by one within the function leads to a sequence of numbers:

```
<?php
function printElement($s, &$i) {
    printf('%d: %s<br />', $i,
    ↪htmlspecialchars($s));
    $i++;
}

$i = 1;
$a = array('one', 'two', 'three', 'four');
```

```
$a = array_walk($a, 'printElement', $i);  
?>
```

Printing Array Elements Using array_walk() and a Counter Passed by Reference (array_walk.php)

Running the preceding code shows the following:

```
0: one  
1: two  
2: three  
3: four
```

Filtering Arrays

```
array_filter($values, 'checkMail')
```

```
<?php  
function checkMail($s) {  
    // ...  
}  
  
$values = array(  
    'valid@email.tld',  
    'invalid@email',  
    'also@i.nvalid',  
    'also@val.id'  
);  
echo implode(', ', array_filter($values,  
    ↪ 'checkMail'));  
?>
```

Filtering Valid Email Addresses (array_filter.php)

Imagine you get a bunch of values—from an HTML form, a file, or a database—and want to select which of these values are actually usable and which are not. You could again call `for`, `foreach`, or `while` and find out what is interesting, or you can let PHP do most of the work. In the latter case, get acquainted with the function `array_filter()`. This one takes two parameters: first, the array to be filtered; and second, a function name (as a string) that checks whether an array element is good. This validation function returns `true` upon success and `false` otherwise. The following is a very simple validation function for email addresses; see Chapter 1, “Manipulating Strings,” for a much better one:

```
function checkMail($s) {
    $ampersand = strpos($s, '@');
    $lastDot = strrpos($s, '.');
    return ($ampersand !== false &&
           $lastDot !== false &&
           $lastDot - $ampersand >= 3);
}
```

Now, the code at the beginning of this phrase calls `array_filter()` so that only (syntactically) valid email addresses are left.

As you would expect, the code just prints out the two valid email addresses.

Getting Random Elements Out of Arrays

```
array_rand($numbers, 6)
```

```
<?php
    for ($i = 1; $i <= 49; $i++) {
        $numbers[] = $i;
    }
    // we could use range() instead, too

    echo implode(' ', array_rand($numbers, 6));
?>
```

Picking Random Elements Out of an Array (array_rand.php)

With `array_rand()`, one or more random elements out of an array are determined by random. This can, for instance, be used to draw some lucky numbers. For instance, the German lottery draws 6 numbers out of 49. The preceding code implements this drawing using PHP and `array_rand()`; see Figure 2.7 for its output. The first parameter for this function is the array; the second (optional) one is the number of elements to be returned.

NOTE: If you do not want to pick random elements but want to randomize the order of elements in the array (for example, when shuffling a deck of cards), use the `shuffle()` function.

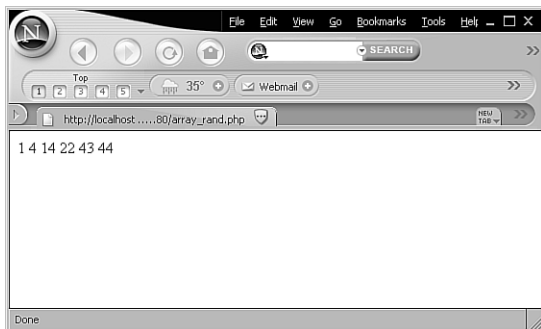


Figure 2.7 Lucky numbers with PHP

Making Objects Behave Like Arrays

`class MyArray implements ArrayAccess, Countable`

```
<?php
class MyArray implements ArrayAccess, Countable {
    private $_data = array();

    /* ArrayAccess interface */
    public function offsetSet($offset, $value) {
        $this->_data[$offset] = $value;
    }

    public function offsetExists($offset) {
        return isset($this->_data[$offset]);
    }

    public function offsetUnset($offset) {
        unset($this->_data[$offset]);
    }
}
```

```

    public function offsetGet($offset) {
        //return (isset($this->_data[$offset])) ?
➔$this->_data[$offset] : null;
        return ($this->offsetExists($offset)) ?
➔$this->_data[$offset] : null;
    }

    /* Countable Interface */
    public function count() {
        return count($this->_data);
    }

}

$a = new MyArray();
$a[0] = 'I';
$a[1] = 'II';
$a[2] = 'III';
$a[3] = 'IV';
for ($i = 0; $i < count($a); $i++) {
    printf('<p>%s</p>', htmlspecialchars($a[$i]));
}
?>

```

Using an Object Like an Array with SPL (splArray.php)

SPL, the Standard PHP Library, is one of the most underrated features of PHP. It basically offers a huge set of interfaces. If you create a class using one of those interfaces, you might use standard PHP functions with your class. The documentation at <http://php.net/spl> is not always as detailed as you would expect, but it does give you some great pointers as to what is possible.

As a simple example, we will create a class (you can find more about various aspects of object-oriented programming [OOP] in Chapter 4) that implements two interfaces:

- **ArrayAccess**—Allows accessing individual elements within the object's collection (in our code, a simple array)
- **Countable**—Allows calling `count()` on an object instance

If you want full array support, you need to implement additional interfaces, including `Iterable` (for iterator support; for example, via `foreach`) and some more. The `ArrayObject` interface aggregates several of the required interfaces, which in turn could even allow advanced features like user-defined sorting.

Index

Symbols and Numerics

- != operator, comparing strings, 26
- [...] special character, in regular expressions, 28
- * (asterisk), in regular expressions, 28
- \ (backslash), in regular expressions, 29
- ^ (caret), in regular expressions, 27
- \$ (dollar sign), in regular expressions, 27
- . (dot), in regular expressions, 28
- == operator, comparing strings, 8-10, 26
- | (pipe), in regular expressions, 28
- + (plus sign), in regular expressions, 28
- ? (question mark), in regular expressions, 27
- \$_COOKIE array, 170
- \$_FILES array uploaded file information, retrieving, 155
- \$_GET array
 - accessing form data, 126-127
 - Graphical Submit buttons, 144
 - mandatory fields, validating, 145
 - prefilling text fields, 131, 133-134
 - preselecting radio buttons, 136
 - preselecting selection lists, 141-142
 - reading in form data, 123
- \$_POST array
 - accessing form data, 126-127
 - Graphical Submit buttons, 144
 - mandatory fields, validating, 145
 - prefilling text fields, 131, 133-134
 - preselecting radio buttons, 136
 - preselecting selection lists, 141-142
- \$_REQUEST array, accessing form data, 126-127
- 401 status code, 198

A

abstract classes, 104-107**access, file access**

PHP built-in class,
221-222

relative paths for,
209-210

security traps,
210-211

addslashes() function,
265

**AES (Advanced
Encryption Standard),**
15

Ajax, 313-316

ajax.php, 314-315

**array() function, creating
arrays, 40**

array_filter() function, 66

array_map() function, 62

array_multisort() function,
54

array_rand() function,
67-68

array_walk() function, 64

arrays

applying effects to
elements, 62-65

converting into
variables, 47-48

converting strings to,
48-49

converting to strings,
49

creating, 40-41

filtering, 65-66

for form data, 125

looping through

associative arrays,
43

nested arrays,
44-47

numerical arrays,
41-43

nesting, 40

random elements,
retrieving, 67-68

sorting

alphabetically,
50-51

associative arrays,
51-52

natural sorting, 57

nested arrays,
53-54

nested associative
arrays, 55-56

QuickSort
algorithm, 58

special language
characters, 59-60

**ASCII codes, protecting
email addresses, 20-22**

asort() function, 52

associative arrays

looping through, 43
sorting, 51-52

**attacks, directory
traversal, 210-211**

Auth package (PEAR),
199
authentication
 session management,
 195-196
 without sessions, 197
autoloading classes,
109-111

B

basename() function, 210
bzclose() function,
225-226
Bzip2 archives, 224-226
bzopen() function,
225-226
bzread() function, 226
bzwrite() function,
225-226

C

check boxes, preselect-
ing, 138
checkdate() function, 82
checksumming strings,
16-18
chr() function, 21
classes
 abstract, 104-107
 autoloading, 109-111
 inheritance, 102-103
 setting up, 100-101
 traits, 120-122

cloning objects, 111-112
closing
 files, 204
 sessions, 187
compareChar() function,
60
comparing strings, 8
compress.bzip2://,
224-226
compress.zlib://, 223
compression
 Bzip2 archives, 224-
 226
 ZIP archives, 223-224
compression streams,
222-224
configureWSDL() method,
307
connecting to
 Firebird, 270-271
 MSSQL, 262
 MySQL, 231-232
 MySQLi, 232
 Oracle, 257
 PostgreSQL, 253
 SQLite, 244-245
connections, FTP servers,
298-300
converting
 arrays
 into variables,
 47-48
 into strings, 49

- data types, 32-33
- strings
 - into dates, 85-86
 - into HTML, 11-13
 - strings to arrays, 48-49
- Cookie HTTP header, 165**
- cookies, 164-166**
 - creating, 167-169
 - deleting, 173
 - domains, setting, 174-175
 - expiry date, setting, 166, 170-172
 - limitations, 164
 - prefilling form data from, 132-134
 - raw cookie data, sending, 169
 - reading out, 170
 - saving
 - language preferences, 180-181
 - multiple values, 177-178
 - saving HTML form data into, 129-131
 - specification, 165
 - testing browser support, 176-177
- cookietest.php, 176-177**
- copy() function, 221**
- copying files, 221**
- count() function, 42-43**

- creating**
 - arrays, 40-41
 - checksums, 16-18
- Cross-Site Scripting (XSS) attacks, removing HTML markup, 11-13**
- crypt() function, 14-15**
- CSV files, 212-215**

D

- data, exchanging with JSON, 316, 319**
- Data Encryption Standard (DES), encrypting strings, 14-15**
- data types, 32-33. See also specific data types**
- databases**
 - Firebird
 - connecting to, 270-271
 - retrieving data from, 273
 - sending SQL statements to, 272
 - MSSQL
 - connecting to, 262
 - retrieving data from, 266
 - sending SQL statements to, 264-265
 - MSSQL/MSDE, retrieving data from, 269

- MySQL
 - connecting to, 231-232
 - prepared SQL statements, 235
 - retrieving data from, 237-239
 - sending SQL statements to, 233-234
- Oracle, connecting to, 257
 - retrieving data from, 261-262
 - sending SQL statements to, 259-260
- PEAR packages, 278
- PostgreSQL
 - connecting to, 253
 - inserting/updating data, 255
 - retrieving data from, 256-257
 - sending SQL statements to, 254
- SQLite
 - connecting to, 244-245
 - retrieving data from, 248-250
 - sending SQL statements to, 247
- date() function, 71-73**
 - epoche values, 81
 - formatting symbols, 71-73
 - localizing dates, 79
 - using text within, 74
- Date_Holidays package (PEAR), 97**
- date_sunrise() function, 86-87**
- date_sunset() function, 86-87**
- date and time, 71. See also date() function**
 - benchmark operations, 88-89
 - converting strings into dates, 85-86
 - difference between two dates, calculating, 93-94
 - for sunrise and sunset, 86-87
 - GMT format, 96-97
 - HTML form selection fields, 90-93
 - leap years, determining, 82
 - localizing dates, 76-79
 - PEAR packages, 97
 - relative dates, calculating, 83
 - sortable time stamps, creating, 84

- specific dates, formatting, 81
- validating dates, 82
- Date package (PEAR), 97**
- DB_DataObject package (PEAR), 278**
- deletecookie.php, 173**
- deleting cookies, 173**
- DES (Data Encryption Standard), encrypting strings, 14-15**
- deserializing objects, 113, 115**
- directory traversal attacks, 210-211**
- dirname() function, 209-210**
- DOM (Document Object Model), 284**
 - reading XML, 284-285
 - writing XML, 285-286
- drivers, PDO (PHP Data Objects), 274-275**

E

- each() function, 42-43**
- email, sending HTML form data via, 153**
- email addresses**
 - protecting using ASCII codes, 20
 - validating, 34-35
- encrypting strings, 14-15**

epoch values

- calculating differences between dates, 93-94
- converting strings into, 85-86
- date() formatting symbol, 73
- specific dates, formatting, 81
- European date formats, 80**
- exchanging data with JSON, 316, 319**
- explode() function, 49**
- expressions**
 - printing, 22-24
 - regular. See regular expressions
- ext/mysql extension, 231**
- Extensible Markup Language. See XML**
- extracting substrings, 19**

F-G

- fclose() function, 204**
- fgetcsv() function, 212-214**
- fgets() function, 206**
- File_Find package (PEAR), 228**
- file() function, 206**
- file_exists() function, 204**
- file_get_contents() function, 205, 298**

- file_put_contents()**
function, 207, 298
- File_SearchReplace**
package (PEAR), 228
- File** package (PEAR), 228
- fileatime()** function, 219
- filegroup()** function, 219
- filemtime()** function, 219
- fileowner()** function, 219
- files**
 - access
 - PHP built-in class, 221-222
 - relative paths for, 209-210
 - security traps, 210-211
 - closing, 204
 - copying, 221
 - CSV files, 212-215
 - INI files, parsing, 216-217
 - locking, 208-209
 - opening, 202-203
 - PEAR packages, 228
 - reading data from, 205-206
 - renaming, 221
 - retrieving file information, 218-219
 - returning with HTTP requests, 227-228
 - unlinking, 221
 - uploading
 - moving uploaded files, 157-158
 - uploaded file information, retrieving, 154-155
 - writing data to, 207
- filesize()** function, 219
- filtering** arrays, 65-66
- Firebird**
 - connecting to, 270-271
 - retrieving data from, 273
 - sending SQL statements to, 272
- flock()** function, 208-209
- fopen()** function, 202-203
- for** loops, numerical arrays, 42
- foreach** loops
 - associative arrays, 43
 - numerical arrays, 41
- formatting** symbols, **strftime()** function, 76-77
- fputcsv()** function, 214-215
- fsocketopen()** function, 297
- ftp-file.php**, 298
- ftp-functions.php**, 300
- ftp_get()** function, 300

**FTP server connections,
298-300**

- `getcookie.php`, 170
- `getCookieData()` function, 130, 179
- `getCookieData.inc.php`, 178-179
- `getFormDataGET()` function, 134
- `getFormDataPOST()` function, 134
- `gettimeofday()` function, 88-89
- global arrays for form data, 125
- `gmdate()` function, 97
- `gmmktime()` function, 97
- `gmstrftime()` function, 97
- GMT time format, 96-97
- Graphical Submit buttons, 144

H

HTML. See also HTML forms

- character codes for email addresses, 21-22
- converting strings into, 11-13
- line breaks within, 14
- tags
 - finding with regular expressions, 31

- removing, 12-13

HTML forms, 123-124

- date selection fields, 90
- form data
 - reading out, 125-127
 - saving into cookies, 129-131
 - sending back to current script, 124-125
 - sending via email, 153
 - writing into files, 151-152, 158
- mandatory fields, validating, 145-146
- prefilling
 - password fields, 131, 134
 - text fields, 131, 134-135
- preselecting
 - check boxes, 138
 - radio buttons, 136-137
 - selection lists, 139-143
- selection lists
 - for date selection, 90-93
 - preselecting, 139-143
 - validating, 147-149

- submission, checking, 128
 - validating mandatory fields, 31-32
 - htmlentities() function, 11-12
 - htmlspecialchars() function, 11-12, 62, 131, 134-135
 - HTML_QuickForm2 package (PEAR), 161
 - HTTP headers, viewing, 165
 - HTTP requests, returning files with, 227-228
 - http-file.php, 295
 - HTTP_Session package (PEAR), 199
 - http-socket.php, 297
 - http_upload package (PEAR), 161
-
- I**
- ibase extension, 270
 - ibase_connect() function, 270-271
 - ibase_connect.php, 270
 - ibase_execute() function, 272
 - ibase_fetch.php, 273
 - ibase_fetch_assoc() function, 273
 - ibase_fetch_object() function, 273
 - ibase_prepare() function, 272
 - ibase_query() function, 272
 - implementing singletons, 116-119
 - implode() function, 49, 206
 - inheritance, 102-103, 108
 - INI files, parsing, 216-217
 - InterBase. See Firebird interfaces, 104-107
 - IP addresses, sorting, 57
 - is_array() function, 32
 - is_bool() function, 32
 - is_dir() function, 219
 - is_executable() function, 219
 - is_file() function, 219
 - is_float() function, 32
 - is_int() function, 32
 - is_link() function, 219
 - is_null() function, 32
 - is_numeric() function, 32-33
 - is_object() function, 32
 - is_readable() function, 219
 - is_string() function, 32
 - is_uploaded_file() function, 219
 - is_writable() function, 219

J-K-L

JavaScript Object

Notation: see *JSON*

JSON, data exchange,
316, 319

krsort() function, 52

ksort() function, 52, 55

leap years, determining,
82

line breaks, 14

list() function, 48

localizing dates, 76-79

locking files, 208-209

looping through

associative arrays, 43

nested arrays, 44-47

numerical arrays,
41-43

M

MD5 (Message Digest
Algorithm 5), 17-18

md5() function, 16-18

MDB2 package (PEAR),
278

MDB_QueryTool package
(PEAR), 278

Microsoft SQL Server. See
MSSQL

Microsoft SQL Server
Express Edition, 262

microtimestamp() func-
tion, 88-89

mktime() function, 81-82

mod_files.sh script, 183

moving uploaded files,
157-158

MSDE

retrieving data from,
269

sending SQL state-
ments to, 264-265

MSSQL (Microsoft SQL
Server)

connecting to, 262

retrieving data from,
266, 269

sending SQL state-
ments to, 264-265

mssql_connect() function,
269

mssql_fetch_assoc() func-
tion, 269

mssql_fetch_object()
function, 269

mssql_fetch_row() func-
tion, 269

mssql_query() function,
264

multiline text fields,
prefilling, 135

multilingual.php, 181

MySQL

connecting to,
231-232

prepared SQL state-
ments, 235

retrieving data from,
237-239

sending SQL statements to, 233-234
mysqli_close() function, 232
mysqli_connect() function, 232
mysqli_fetch_assoc() function, 237
mysqli_fetch_object() function, 237
mysqli_fetch_row() function, 238
mysqli_insert_id() function, 240
mysqli_prepare() function, 235-236
mysqli_query() function, 233
mysqli_real_escape_string() function, 233
mysqli_stmt_execute() function, 236
mysql_connect() function, 231-232

N

namespaces, 118
natcasesort() function, 57
natsort() function, 57
Natural Order String Comparison project, 57
natural sorting, 57
nested arrays
 looping through, 44-47

 printing, 44, 47
 sorting, 53-56
nesting arrays, 40
Net_FTP package (PEAR), 319
Net_IMAP package (PEAR), 319
Net_Socket() package (PEAR), 319
nl2br() function, 14
numeric data types,
 converting, 32-33
numerical arrays
 looping through, 41-43
 sorting, 50-51
NuSOAP
 consuming Web Services, 309
 creating Web Services, 304-305
 generating WSDL, 306-308

O

objects
 cloning, 111-112
 deserializing, 113-115
 serializing, 113-115
oci_bind_by_name() function, 259
oci_connect() function, 257
oci_connect.php, 257

oci_execute() function, 259

oci_execute.php, 259

oci_fetch.php, 261

oci_fetch_all() function, 261

oci_fetch_assoc() function, 261

oci_fetch_object() function, 261

oci_fetch_row() function, 261

oci_parse() function, 259

OOP, 99

opening files, 202-203

Oracle

- connecting to, 257
- retrieving data from, 261-262
- sending SQL statements to, 259-260

ord() function, 21

P

packages. See PEAR packages

parse_ini_file() function, 216-217

parsing

- INI files, 216-217
- XML with SAX, 280-282

password fields, prefilling, 131, 134

passwords

- encrypting, 14-15
- validating, 9-10

PCRE (Perl-compatible regular expressions), 27-30

PDO (PHP Data Objects), 274-277

pdo_execute.php, 276

pdo_fetch.php, 277

PEAR (PHP Extension and Application Repository), 319. See also PEAR Packages

PEAR packages

- Auth, 199
- Date, 97
- Date_Holidays, 97
- DB_DataObject, 278
- File, 228
- File_Find, 228
- File_SearchReplace, 228
- HTML_QuickForm2, 161
- HTTP_Session, 199
- HTTP_Upload, 161
- MDB2, 278
- MDB_QueryTool, 278
- Net_FTP, 319
- Net_IMAP, 319
- Net_Socket, 319
- Stream_Var, 228
- XML_Beautifier, 293

- XML_DTD, 293
- XML_Parser2, 293
- XML_Serializer, 293
- XML_Util, 293
- pg_connect() function, 253**
- pg_connect.php, 253**
- pg_escape_string(), 254**
- pg_fetch_all() function, 256**
- pg_fetch_assoc() function, 256**
- pg_fetch_object() function, 256**
- pg_fetch_row() function, 256**
- pg_insert() function, 255**
- pg_insert.php, 255**
- pg_last_oid() function, 254**
- pg_query() function, 254, 256**
- pg_select() function, 257**
- pg_update() function, 255**
- PHP**
 - secure pages, 197
 - streams, 222-224
- PHP Data Objects. See PDO**
- PHP5-SOAP**
 - consuming Web Services, 312-313
 - creating Web Services, 310
 - generating WSDL, 310-311
- phpMyAdmin, 230**
- Pool, Martin, 57**
- PostgreSQL**
 - connecting to, 253
 - inserting/updating data, 255
 - retrieving data from, 256-257
 - sending SQL statements to, 254
- prefilling HTML forms**
 - password fields, 131, 134
 - text fields, 131, 134-135
- preg_match() function, 29-30**
- preg_match_all() function, 29-30**
- preg_replace() function, 35**
- prepared statements, with MySQL, 235**
- preselecting**
 - check boxes, 138
 - radio buttons, 136-137
 - selection lists, 139-143
- preventing inheritance, 108**
- printf() function, 22-24**
- print_r() function, 44, 47**

Q-R

Quicksort algorithm, 58

radio buttons, preselecting, 136-137

randomizing array elements, 67

raw cookie data, sending, 169

reading

data from files, 205-206

sessions, 186

XML, 284-285

reading out cookies, 170

reading out form data (HTML forms), 125-127

regular expressions, 27-29

finding tags with, 31

PCRE, 29-30

searching and replacing within strings, 35-36

special characters in, 27-29

relative dates, calculating, 83

relative paths for file access, 209-210

relaxNG, validating XML against, 291

rename() function, 221

renaming files, 221

requests (HTTP), returning files, 227-228

REST (Representational State), 302

retrieving query data

from Firebird, 273

from MySQL, 237-239

from Oracle, 261-262

from PostgreSQL, 256-257

from SQLite, 248-250

via PDO, 277

rsort() function, 51

S

sajax_show_javascript() function, 314

saveLanguage.inc.php, 180-181

saving HTML form data into cookies, 129-131

SAX (Simple API for XML), 280-282

scanning formatted strings, 24-25

schemas, validating XML against, 291-292

searching

for substrings, 26

within strings, 35-36

security, file access, 210-211

selection lists

for date selection, 90-93

preselecting, 139-143

validating, 147-149

- serialize() function, 132
- serializing objects, 113-115
- serverstatus.php, 301
- session.auto_start directive, 185
- session.save_path directive, 183
- session.user_trans_sid directive, 184
- session_authentication.inc.php, 195
- session_destroy() function, 187
- session_readwrite.php, 186
- session_regenerate_id() function, 188
- session_regenerate_id.php, 188
- session_start() function, 185
- session_start.php, 185
- sessions, 182-183
 - activating, 185
 - closing, 187
 - custom management, implementing, 189-190, 193-194
 - ID, changing, 188-189
 - maintaining state, 184
 - PEAR packages, 199
 - reading and writing, 186
 - storing, 183
 - user authentication, checking, 195-196
- Set-Cookie HTTP header, 165
- setcookie() function, 167-169
- setcookie-domain.php, 174-175
- setcookie-expiry.php, 170-171
- setcookie-specific.php, 171-172
- setcookie.php, 167
- setlocale() function, 76-77
- setrawcookie() function, 169
- setting up classes, 100-101
- SHA1 (Secure Hash Algorithm 1), 17-18
- sha1() function, 16-18
- shuffle() function, 67
- SimpleXML extension, 289
- simplexml_import_dom() function, 289
- simplexml_load_file() function, 289
- simplexml_load_string() function, 289
- singletons, implementing, 116-119

SOAP, 303-304**soap-nusoap-server.php, 304****soap-php5-server.php, 310****sort() function**

associative arrays, 52

nested arrays, 54

numerical arrays, 50-51

sortable time stamps, creating, 84**sorting**

arrays

alphabetically, 50-51

associative arrays, 51-52

natural sorting, 57

nested arrays, 53-54

nested associative arrays, 55-56

QuickSort algorithm, 58

special language characters, 59-60

IP addresses, 57

special characters in regular expressions, 27-29**sprintf() function, 24****SQL statements**

prepared statements with MySQL, 235

query data, retrieving from Firebird, 273

from MSSQL, 266

from MSSQL/MSDE, 269

from MySQL, 237-239

from Oracle, 261-262

from PostgreSQL, 256-257

from SQLite, 248-250

via PDO, 277

sending

to Firebird, 272

to MSSQL, 264-265

to MySQL, 233-234

to Oracle, 259-260

to PostgreSQL, 254

to SQLite, 247

via PDO, 276

SQLite

connecting to, 244-245

retrieving data from, 248-250

sending SQL statements to, 247

sqlite_escape_string() function, 247**sqlite_exec() function, 247-248****sqlite_exec.php, 247****sqlite_fetch_all() function, 249-250**

- sqlite_fetch_array() function, 249-250
- sqlite_fetch_object() function, 249-250
- sqlite_last_insert_rowid() function, 247
- sqlite_open() function, 245
- sqlite_open.php, 244-245
- sqlite_query() function, 248
- sqlsrv_fetch_array() function, 266
- sqlsrv_fetch_object() function, 266
- sscanf() function, 24
- strcasecmp() function, 8-10
- strcmp() function, 8-9
- streams, 222-224
 - PEAR packages, 228
 - returning contents, 205
- Stream_Var package (PEAR), 228
- strftime() function, 76-78
- strings
 - checksumming, 16-18
 - comparing, 8
 - converting
 - from arrays, 49
 - to arrays, 48-49
 - to dates, 85-86
 - to HTML, 11-13
 - encrypting, 14-15
 - extracting substrings, 19
 - in date() function, 74
 - in URLs, 12
 - printing, 22-24
 - regular expressions, 27-31
 - scanning formatted strings, 24-25
 - searching and replacing within, 35-36
 - searching for substrings, 26
 - validating
 - data types, 32-33
 - email addresses, 34-35
 - mandatory form field input, 31-32
 - passwords, 9-10
 - usernames, 9-10
 - variables, getting information about, 25-26
- strip_tags() function, 12-13
- strnatcmp() function, 57, 60
- strops() function, 26
- strtolower() function, 10
- strtotime() function, 85-86
- strtoupper() function, 10
- str_replace() function, 36

substr() function, 19

substrings

extracting, 19

searching for, 26

sunrise and sunset time, determining, 86-87

superglobal arrays, accessing form data, 126

symbols, date() function, 71-73

T

tags (HTML)

finding with regular expressions, 31

removing, 12-13

text

fields, prefilling, 131, 134-135

in date() function, 74

time. See date and time time and date, date formats, 80

time stamps

benchmarks, 88-89

calculating relative dates, 83

converting strings into, 85-86

GMT format, 97

sortable time stamps, creating, 83-84

traits, 120-122

transforming XML, 290

trim() function, 32

U

United Kingdom date formats, 80

United States date formats, 80

unlink() function, 221

unlinking files, 221

unserialize() function, 132

uploading files

moving uploaded files, 157-158

uploaded file information, retrieving, 154-155

urlencode() function, 12

URLs, preparing strings for use in, 12

usernames, validating, 9-10

usort() function, 58

V

validating

data types, 32-33

dates, 82

email addresses, 34-35

HTML forms

mandatory fields, 31-32, 145-146

- selection lists, 147-149
 - passwords, 9-10
 - usernames, 9-10
 - XML, 291-292
 - variables**
 - checking data type of, 32-33
 - converting arrays into, 47-48
 - information about, getting, 25-26
 - printing, 22-24
 - var_dump() function, 25**
 - var_export() function, 26**
 - vprintf() function, 24**
 - vsprintf() function, 24**
-
- W**
- web forms. See HTML forms**
 - web servers**
 - FTP server connections, 298-300
 - HTTP server connections, 295-298
 - status, checking, 301
 - Web Services, 302-304**
 - consuming
 - with NuSOAP, 309
 - with PHP5-SOAP, 312-313
 - creating
 - with NuSOAP, 304-305
 - with PHP5-SOAP, 310
 - WSDL, 303
 - generating with NuSOAP, 306-308
 - generating with PHP5-SOAP, 310-311
 - while loops**
 - associative arrays, 43
 - numerical arrays, 42
 - writing**
 - HTML form data into files, 151-152, 158
 - sessions, 186
 - XML, 285-286
 - writing data to files, 207**
 - WSDL (Web Services Description Language), 303**
 - generating with NuSOAP, 306-308
 - generating with PHP5-SOAP, 310-311
 - wsdl-nusoap-client.php, 308**
 - wsdl-nusoap-server.php, 306**
 - wsdl-php5-client.php, 312**
 - wsdl-php5-server.php, 312**

X-Y

XML, 279-280

parsing with SAX,
280-282

PEAR packages, 293

reading PHP 4, 284-
285

SimpleXML extension,
289

validating, 291-292

writing, 285-286

XSLT (XSL
Transformations),
290

XML-RPC, 302

**XML_Beautifier package
(PEAR), 293**

**XML_DTD package
(PEAR), 293**

**XML_Parser2 package
(PEAR), 293**

**xml_parser_create() func-
tion, 281**

**xml_parser_set_option()
function, 281**

**XML_Serialize package
(PEAR), 293**

**xml_set_character_data_
handler() function, 281**

**xml_set_element_han-
dler() function, 281**

**XML_Util package
(PEAR), 293**

**XSLT (XSL
Transformations), 290**

**XSS (Cross-Site Scripting)
attacks**

HTML markup, remov-
ing, 11-13

Z

ZIP files

built-in PHP functions,
223-224

compress.zlib://, 223

**zip_entry_open() function,
223**

**zip_entry_read() function,
223**

zip_open() function, 223

zip_read() function, 223

ZZIPLib library, 223