



Read Less-Learn More®

Adam McDaniel

Perl and Apache



Sample code available on
the companion Web site

*Your visual blueprint™ for
developing dynamic Web content*

Perl and Apache

*Your visual blueprint™ for developing
dynamic Web content*



by Adam McDaniel



Wiley Publishing, Inc.

Perl and Apache: Your visual blueprint™ for developing dynamic Web content

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256

www.wiley.com

Published simultaneously in Canada

Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, 201-748-6011, fax 201-748-6008, or online at www.wiley.com/go/permissions.

Library of Congress Control Number: 2010934753

ISBN: 978-0-470-55680-1

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Trademark Acknowledgments

Wiley, the Wiley Publishing logo, Visual, the Visual logo, Visual Blueprint, Read Less - Learn More and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

FOR PURPOSES OF ILLUSTRATING THE CONCEPTS AND TECHNIQUES DESCRIBED IN THIS BOOK, THE AUTHOR HAS CREATED VARIOUS NAMES, COMPANY NAMES, MAILING, E-MAIL AND INTERNET ADDRESSES, PHONE AND FAX NUMBERS AND SIMILAR INFORMATION, ALL OF WHICH ARE FICTITIOUS. ANY RESEMBLANCE OF THESE FICTITIOUS NAMES, ADDRESSES, PHONE AND FAX NUMBERS AND SIMILAR INFORMATION TO ANY ACTUAL PERSON, COMPANY AND/OR ORGANIZATION IS UNINTENTIONAL AND PURELY COINCIDENTAL.

Contact Us

For general information on our other products and services please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For technical support please visit www.wiley.com/techsupport.

The Diwan-I-Khas of the palace complex at Fatehpur Sikri

The history of this intriguing pavilion is almost as enigmatic as the structure itself. Dating to the 17th century, this private audience hall is remarkable for its richly carved central pillar, unique in Mughal architecture. One school of thought holds that the design of the building and its stone centerpiece may reflect some Hindu mandala in which the central column represents the axis of the world. As such, it conferred superior status upon the emperor who received visiting dignitaries while seated at its base.



Explore India's countless architectural treasures in *Frommer's India*, 4th Edition (ISBN 978-0-470-55610-8) available wherever books are sold or at www.Frommers.com.



Sales

Contact Wiley
at (877) 762-2974
or (317) 572-4002.

Credits

Acquisitions Editor

Aaron Black

Project Editor

Jade Williams

Technical Editor

Allen Wyatt

Copy Editor

Marylouise Wiack

Editorial Director

Robyn Siesky

Editorial Manager

Rosemarie Graham

Business Manager

Amy Knies

Senior Marketing Manager

Sandy Smith

Vice President and Executive Group Publisher

Richard Swadley

Vice President and Executive Publisher

Barry Pruett

Project Coordinator

Patrick Redmond

Graphics and Production Specialists

Andrea Hornberger

Jennifer Mayberry

Quality Control Technician

Jessica Kramer

Proofreading

Tricia Liebig

Indexing

Potomac Indexing, LLC

Media Development Project Manager

Laura Moss

Media Development Assistant Project Manager

Jenny Swisher

Screen Artist

Ana Carrillo

Jill Proll

Ron Terry

Illustrator

Cheryl Grubbs

About the Author

Adam McDaniel has been designing, developing, modifying, and maintaining computer programs of one language or another since 1993, and has been an active proponent of Perl since being introduced to the language in 1998. In early 1999, Adam led a team of developers implementing an E-Commerce fulfillment engine written entirely in Perl for a virtual shopping mall. Afterwards, he worked for Hitachi ID Systems for over 8 years, during which he designed and implemented security recommendations and software for various Fortune 500 companies across the United States and Europe.

Always interested in new technologies and architectures, development credits include an open-source offline HTML reader for the Palm OS platform, contributions to the Linux Kernel, plus countless utility and speciality programs. In 2006, Adam produced the Array.org Netbook Kernel software download and Web site, allowing users to download an optimized build of the Linux kernel, specific for the Ubuntu Linux distribution. This site, implemented using Perl and Apache, became hugely popular resulting in millions unique visitors in just a few months. It actually prompted him to move away from security architecture and design and into a new field: Linux distribution architecture. Today, Adam works as the Lead OS Architect for Jolicloud, a Paris-based Linux distribution that specializes in a cloud-based user interface designed for netbooks, tablets, and other portable computers.

Author's Acknowledgments

This book is actually the product of many significant people, projects, and events, without all of which, this project would never have been possible.

First and foremost, I must thank my wife, Shauna, for her un-ending patience, understanding, encouragement, and love, both silent and vocal, which she happily supplied while I toiled away endlessly on this project. I must also thank my editors at Wiley Publishing, especially Aaron Black and Jade Williams, for their expert guidance, advice, and patience, despite their occasional prodding over e-mail.

This book could not be possible without hundreds of thousands of developers who have dedicated their time and expertise to open-source software. Projects like the Linux Kernel, Perl, Apache, and everything in-between serves as an excellent model of design, efficiency, and dedication to people like me and other technology enthusiasts.

Finally, regarding significant events, I have to thank our unpredictable Canadian winter weather. In January 2008, we experienced seven days of -40 degree weather; this caused a water pipe to burst and flood my basement with 18 inches of water while I was away from home. Had my basement not flooded, I would never have replaced an old waterlogged laptop with a brand new EeePC netbook through the insurance claim. Without that, I would have never become interested in netbook hardware, nor in customizing the Linux kernel for that hardware. And I certainly would have never created my Array.org Netbook Kernel Web site, through which Aaron Black would never have contacted me, and this book would never have come into existence.

Dedication

This book is dedicated to my father, William McDaniel, who knew about this project, but never got the chance to see the final result.

How to Use This Book

Who This Book Is For

This book is for advanced computer users who want to take their knowledge of this particular technology or software application to the next level.

The Conventions in This Book

1 Steps

This book uses a step-by-step format to guide you easily through each task. Numbered steps are actions you must do; bulleted steps clarify a point, step, or optional feature; and indented steps give you the result.

2 Notes

Notes give additional information — special conditions that may occur during an operation, a situation that you want to avoid, or a cross reference to a related area of the book.

3 Icons and Buttons

Icons and buttons show you exactly what you need to click to perform a step.

4 Extra or Apply It

An Extra section provides additional information about the preceding task — insider information and tips for ease and efficiency. An Apply It section takes the code from the preceding task one step further and allows you to take full advantage of it.

5 Bold

Bold type shows text or numbers you must type.

6 Italics

Italic type introduces and defines a new term.

7 Courier Font

Courier font indicates the use of scripting language code such as statements, operators, or functions, and code such as objects, methods, or properties.

Call a Module's Subroutines as Methods

Before a Perl script can take advantage of the subroutines contained within a module, a module reference needs to be established. Just like an array or hash reference, a *module reference* is a scalar value that holds the address of a module. That scalar then acts as a handle to the module's contents, including its variables, methods, and shared *\$self* hash reference. Naturally, a single script may have many module instances running in parallel, even multiple instances of the same module, if required. It is the individual script that holds the module reference that keeps everything organized. It is that same scalar that is used to access the module's subroutines as methods. This happens through an arrow

Call a Module's Subroutines as Methods

- 1 Open a new Perl script in a text editor.
- 2 Identify a Perl module that you want to load.
- 3 Type `use Module;`
- 4 Type `my $handle = Module->new();` to declare a new scalar, initialize the module, and store the module reference.
- 5 Type `$handle->Function();` to execute one of the exported functions in the module.
- 6 Note: Even though `Function` is a subroutine, it is not correct to provide it with an ampersand.

```
use Module;
my $h = Module->new();
$h->newModule();
You can even use the handle to access the same $self variable used within the actual module. In the exact same way that subroutines are executed as methods, variables can be manipulated using the handle.
$h->{ KEY } = VALUE;
Regardless of whether your module utilizes new anywhere, the script can use it to store additional data, just like any type of complex hash reference. When the program ends, all active module references are automatically released from memory. It is possible to manually destroy a module by calling undef on the module reference.
```

Extra

It is possible to pass additional variables to the module's functions, just like any other subroutine. The only difference is that the `$self` variable is the first parameter in the `@_array`. If your function allows for other parameters, then you can write them as

```
sub Function {
    my ( $self, $param1, $param2 ) = @_;
    [...]
}
```

When `Function` is executed within the context of a Perl script, the `$self` parameter is implicitly passed to the method.

```
$self->Function( $param1, $param2 );
```

However, if two methods within a module need to execute each other, they should be treated like regular subroutines, not methods. For example, if `Function` has to call the method `Calculate`, then it must specify `$self` explicitly.

```
sub Function {
    my $self = shift;
    $self->Calculate( $self, ... );
}
```

Chapter 8: Using Perl References and Modules



TABLE OF CONTENTS

HOW TO USE THIS BOOK XII

1 INTRODUCING PERL AND APACHE WEB SITE DEVELOPMENT 2

Introducing Apache and Perl.....	2
Introducing the Common Gateway Interface.....	4
Understanding CGI from the End-User's Point of View.....	6
Understanding CGI from the Web Browser's Point of View.....	8
Understanding CGI from the Web Server's Point of View.....	10
Understanding CGI from the CGI Program's Point of View.....	12
Compare Perl to Other CGI Languages.....	14
Compare Apache to Other Web Servers.....	16
Developing Your Web Site.....	18
Find Perl- and Apache-Friendly Hosting Providers.....	20
Find Help Developing CGI Programs.....	22

2 INSTALLING PERL ON WINDOWS 24

Introducing ActivePerl for Windows.....	24
Introducing Strawberry Perl for Windows.....	25
Download ActivePerl for Windows.....	26
Install ActivePerl for Windows.....	28
Download Strawberry Perl for Windows.....	30
Install Strawberry Perl for Windows.....	32

3 INSTALLING PERL ON LINUX 34

Install Perl for Debian/Ubuntu Linux.....	34
Install Perl for Red Hat Linux.....	35
Download ActivePerl for Linux or Unix.....	36
Install ActivePerl for Linux or Unix.....	38

4 INSTALLING APACHE ON WINDOWS 40

Download Apache for Windows.....	40
Install Apache for Windows.....	42
Configure Apache on Windows.....	44
Start and Stop the Apache Service on Windows.....	46

5 INSTALLING APACHE ON LINUX 48

Install Apache for Debian/Ubuntu Linux48

Install Apache for Red Hat Linux.....49

Configure Apache on Linux50

Start and Stop the Apache Service on Linux52

6 INTRODUCING THE FUNDAMENTALS OF PERL 54

Understanding Perl Syntax54

Understanding the Anatomy of a Perl Script.....57

Create a New Perl Script.....58

Print Output to the Screen60

Execute a Perl Script62

Introducing Perl Scalars64

Store Data into Scalars.....66

Retrieve Data from Scalars.....67

Introducing Perl Arrays.....68

Store Data into Arrays70

Retrieve Data from Arrays71

Introducing Perl Hashes.....72

Store Data into Hashes74

Retrieve Data from Hashes75

7 BUILDING AN INTERACTIVE PERL SCRIPT 76

Introducing Perl Conditions.....76

Introducing Perl Operators78

Control Program Flow with if, elsif, else80

Introducing Perl Loops.....82

Loop Program Flow with foreach, while.....84

Introducing Perl Subroutines86

Organize Program Code with Subroutines.....88

Manipulate Variables in Subroutines90

8 USING PERL REFERENCES AND MODULES. 92

Introducing References92

Understanding Compound Data Structures94

Build an Array or Hash Reference96



TABLE OF CONTENTS

Deconstruct a Reference.....	98
Nest Variable Types with References	100
Introducing Perl Modules	102
Create a New Module	104
Call a Module's Subroutines as Methods.....	106

9 INSTALLING THIRD-PARTY PERL MODULES 108

Introducing CPAN	108
Configure CPAN	110
Search for Perl Modules with CPAN	111
Install Perl Modules with CPAN	112
Introducing ActivePerl Perl Package Manager	114
Configure ActivePerl PPM	116
Search for Perl Modules with ActivePerl PPM	118
Install Perl Modules with ActivePerl PPM	119
Search for Perl Modules in Debian/Ubuntu Linux.....	120
Install Perl Modules in Debian/Ubuntu Linux.....	121
Search for Perl Modules in Red Hat Linux	122
Install Perl Modules in Red Hat Linux	123
Search for and Download Perl Modules Manually	124
Build and Install Perl Modules Manually	126

10 CONFIGURING APACHE TO EXECUTE PERL . . . 128

Introducing the Apache CGI Handler.....	128
Create a User Directory for Apache in Windows	130
Create a User Directory for Apache in Linux.....	132
Enable the Apache CGI Module and Handler.....	134
Configure a Directory to Use the CGI Handler	136
Understanding the Apache Logs	138
Configure the Apache Logs	139
Read the Apache Logs	140
Forward Perl Activity into the Apache Logs.....	141

11 INTRODUCING DO-IT-YOURSELF PERL/CGI INTERACTION 142

Create an HTML Form.....	142
Read HTTP GET/POST Parameters.....	144
Introducing Cookies	146

Store HTTP Cookies	148
Retrieve HTTP Cookies.....	150
Send an E-Mail Message.....	152

12 USING PERL'S BUILT-IN CGI LIBRARY 154

Introducing the Built-In CGI Library	154
Import the CGI Library as an Object.....	156
Import the CGI Library's Routines as Functions.....	157
Read HTTP GET/POST Parameters with the CGI Library	158
Store HTTP Cookies with the CGI Library.....	160
Retrieve HTTP Cookies with the CGI Library.....	162
Return Useful Error Messages with CGI::Carp	164

13 SEPARATING HTML CODE FROM PERL CODE . . . 166

Understanding the Benefits of Separating HTML from Perl	166
Introducing the Perl HTML::Template Module.....	168
Understanding the Structure of an HTML::Template File	170
Create a New Template File.....	172
Import the HTML::Template Module	174
Display Data with TMPL_VAR	176
Control Template Content with TMPL_IF, TMPL_ELSE.....	178
Repeat Template Content with TMPL_LOOP.....	180
Nest Templates with TMPL_INCLUDE	182
Create an HTML::Template Header and Footer.....	184
Create an HTML::Template Toolbar.....	185
Link the Header, Toolbar, and Footer with Dynamic Perl Content	186
Extend HTML::Template to Non-HTML Formats.....	188

14 ADDING DYNAMIC CONTENT WITH SERVER-SIDE INCLUDES (SSI). 190

Introducing Server-Side Includes	190
Enable the Apache SSI Module and Output Filter.....	192
Configure a Directory to Use SSI.....	194
Understanding SSI Elements.....	196
Import Files with SSI.....	198
Execute Programs with SSI.....	199
Set Variables within SSI.....	200
Retrieve Variables with SSI.....	201



TABLE OF CONTENTS

Use Conditional Expressions with SSI.....	202
Display File Statistics with SSI.....	204
Link the Header, Toolbar, and Footer with Static HTML Content.....	206

15 AUTHENTICATING A USER SESSION 208

Understanding Apache User Authentication	208
Secure a Directory Path with Apache	210
Use an Authentication Password File.....	212
Require Only Authorized Users.....	214
Understanding User Authentication in Perl.....	216
Create a Perl Authentication Module.....	218
Access a User's Database.....	220
Store User Credentials in a User's Database.....	222
Check for Session Authorization (Step 1)	224
Display a Login Prompt (Step 2)	226
Validate a User's Credentials (Step 3)	228
Authorize a User's Session (Step 4)	230
Restrict Access to a CGI Script.....	232
Terminate a User Session	234

16 INTERFACING YOUR WEB SITE WITH FACEBOOK 236

Register Your Web Site as a Facebook Application	236
Add a Facebook Social Plugin to Your Web Site	238
Enable Facebook Connect on Your Web Site	240
Understanding the Facebook Canvas Feature for Applications.....	244
Create a Facebook Application with Perl	246

17 INTERFACING WITH THE TWITTER API USING PERL. 248

Introducing the Twitter APIs.....	248
Introducing the Perl Twitter Modules.....	250
Register a New Twitter Application.....	252
Authenticate to Twitter Using OAuth.....	254
Create a MyTwitter Perl Module That Inherits Net::Twitter	258
Post a Twitter Status Update.....	260
Retrieve a Twitter Timeline.....	261
Retrieve a List of Twitter Users you Follow	262

Retrieve a List of Twitter Followers	263
Search for Content Using the Twitter Search API	264
Use the Twitter @Anywhere JavaScript API	266
Follow Real-Time Activity with the Twitter Streaming API	268

18 CREATING DYNAMIC IMAGES WITH PERL 270

Accept a File for Upload	270
Open an Image with Image::Magick	272
Resize or Crop an Image with Image::Magick	273
Manipulate an Image with Image::Magick	274
Save an Image to Disk	275
Display a Dynamic Image to the Browser	276
Implement an Image Captcha Test	278
Produce an Image Gallery	280

19 FACILITATING DYNAMIC AJAX CALLS WITH PERL 284

Introducing AJAX	284
Introducing CGI::Ajax	286
Add CGI::Ajax into Your Perl CGI Scripts	288
Call Perl Subroutines Through JavaScript	290
Call JavaScript Through Perl Subroutines	292
Enable Debug Mode in CGI::Ajax	294
Integrate Perl and XML	296
Integrate Perl and JSON	297

20 PROCESSING CREDIT CARD TRANSACTIONS WITH PERL. 298

Introducing PayPal	298
Sign Up for a PayPal Sandbox Account	300
Create Buyer and Seller Sandbox Accounts	302
Retrieve Your Seller's Sandbox API Credentials	303
Use Business::PayPal::NVP to Connect to PayPal	304
Process a Credit Card Payment with PayPal	306
Use the PayPal Express Checkout API	308
Search Your PayPal Transaction History	312
View a PayPal Transaction's Details	313
Refund a PayPal Transaction	314



TABLE OF CONTENTS

21 ACCESSING A BACK-END MYSQL DATABASE WITH PERL 316

Introducing the MySQL Database.....	316
Understanding the SQL Syntax.....	318
Download MySQL for Windows	320
Install MySQL for Windows	322
Install MySQL for Debian/Ubuntu Linux.....	324
Install MySQL for Red Hat Linux	325
Introducing the Perl DBI Library	326
Connect to a MySQL Database with the DBI Library	328
Retrieve SQL Data Using the DBI Library.....	330
Display SQL Data Through HTML::Template.....	332
Change SQL Data Using the DBI Library.....	334

22 SECURING DYNAMIC WEB SITES 336

Understanding TLS/SSL Encryption.....	336
Create a Private SSL Key.....	338
Generate an SSL Certificate Signing Request.....	339
Sign Your Own CSR to Create a Test SSL Certificate	340
Submit Your CSR to Be Signed by a Certificate Authority	341
Configure Apache to Use TLS/SSL	342
Understanding Security in Perl CGI Development.....	346
Limit CGI Access in Apache	348
Identify Unusual Activity on Your Web Site.....	350
Sanitize User Content in Perl CGI.....	352
Validate User Content in Perl CGI.....	354

23 SPEEDING UP DYNAMIC WEB SITES 356

Introducing the Apache mod_perl Module.....	356
Install the Apache mod_perl Module for Windows.....	358
Install the Apache mod_perl Module for Linux	359
Configure the Apache mod_perl Module.....	360
Understanding mod_perl's Caveats	362

APPENDIX A: PERL REFERENCE 364

Access Perl Documentation	364
Execute Perl on the Command-Line	367

Available Built-In Perl Functions368

Using Perl Pre-Defined Variables376

Perl Operators380

Perl Regular Expressions384

**APPENDIX B: APACHE CONFIGURE AND MODULE
REFERENCE. 386**

Apache Run-Time Configuration Directives386

Apache Base Modules and Directives.....391

Apache Authentication and Authorization Modules and Directives398

Apache Extended Modules and Directives.....404

APPENDIX C: USEFUL PERL MODULES 418

Useful Perl Modules.....418

Introducing Apache and Perl

Since the inception of the World Wide Web in 1989, users, academics, and professionals have been inspired by this new canvas to present information over the Internet. The jump from a text-based interface to a graphical interface would capture the world's imagination on presenting information to the masses.

Content-owners can now store information in a series of files on a server, with end-users accessing that data at their convenience. In the earlier days of the Internet, the server-side information was stored as simple static text files and images, meaning that files were only changed when someone manually made a change and uploaded the new file to the server. As a result, most Web sites did not change very often. With the introduction of the Common Gateway Interface (CGI), Web sites could now use programs in place of static files to dynamically create on-demand content that was unique for every user.

Anyone wanting to participate on the graphical Internet requires a client-side program, called a *Web browser*. This program is installed onto a local workstation, and requires an outgoing connection to the public Internet. The browser establishes a communication link through the network to a server-side counterpart, called a *Web*

server, submits a request, and waits for a response. It is the server's job to interpret the request being made, assess the requester's credentials, open the file or execute a program using CGI, and transmit the results back to the user. Once the browser receives the data, it must decode the transfer and render a graphical representation of the text and images so that the user can interpret the information.

Many Web browser programs are freely available for download, depending on the user's choice of operating system. Popular options include Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, and Apple Safari. For the content-owner, a Web site is delivered to the user's browser by various Web server programs, like Apache HTTP Server or Microsoft Internet Information Service (IIS). Again, the options available depend on the choice of operating system on the server.

The program that utilizes CGI typically runs on the same computer as the Web server. There are multiple languages available today that can interact with CGI, including PHP, Java, and even C and C++, but this book focuses on the Practical Extraction and Report Language, more commonly known as Perl.

The Apache HTTP Server

The Apache HTTP Server, widely viewed today as the *de facto* Web server for Unix and Windows platforms, handles more than 90 percent of World Wide Web traffic.

A History of Apache

First released in 1995, Apache evolved from the remains of the now defunct NCSA HTTPd program, which was the first Web server created to support the Hypertext Transfer Protocol, or HTTP.

Early Web servers were only capable of relaying static content directly from files stored on the Web server's hard drive. Eventually, the CGI protocol was standardized and support was added to Apache.

Versions of Apache

Because Apache has been in development for many years, new versions are constantly being released as *major version milestones*. The latest major release, Apache 2.2, supports a wide range of configuration features, performance enhancements, and third-party modules.

Earlier releases, such as Apache 2.0 and 1.3, are still available and supported. However, you should only consider using an older version if you have a specific reason to do so. If you are just starting out with Apache, use the latest stable release available, Apache 2.2.

The End-User Experience

To provide a pleasurable and esthetically pleasing Web site, the Web site author composes HTML content that describes the site's text, images, and layout. Early Web browsers lacked many of today's client-side technologies that are used to create dynamic content, such as Flash, Java, and JavaScript. Instead, they relied on the Web server to provide the entire end-user experience.

Because today's Internet has many options for dynamic content, a Web site author may choose any type of technology that is available: client-side, server-side, or even both. Intelligently mixing technologies that complement each other can create a memorable site that your users will want to visit again.

The Perl Programming Language

Following its own development path, Perl matured independently from Apache as a multi-purpose scripting language for Unix. The Perl programming language uses a syntax structure that is very similar to C in design, yet free and malleable in implementation. The Perl language is classified as a *third-generation*, or *high-level*, programming language; the programmer does not need to worry about complex memory allocation or architecture-specific, low-level CPU interaction. The program file source code, or *script*, is scanned and interpreted, not compiled, by the Perl interpreter at run time.

A History of Perl

When Perl was originally developed, it was not intended to help Web servers deliver dynamic content. Perl 1.0 was released in 1987 as a tool to read, print, sort, report, and interpret large amounts of data efficiently; it quickly became a useful tool for programmers and system administrators. Today's generation, Perl 5, is widely viewed as the most common and most stable version available. The next generation of the language, Perl 6, is currently under development and available as an experimental release.

Versions of Perl

Like Apache, Perl has experienced several major release milestones. The latest stable release, Perl 5.12, is a staple program on virtually all recent releases of Unix-based operating system distributions.

The latest experimental release of Perl 6 introduces major changes to Perl 5 syntax and internals. Because both generations will remain in active development, there is no need to switch your programming focus to Perl 6 after you learn Perl 5, however converting a script is not complicated.

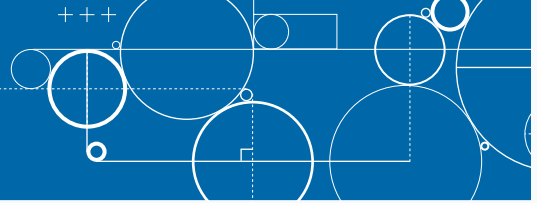
The New, Dynamic Internet

By 1993, Perl was being used in tandem with Web servers to supply content over the Web by executing a program. With the new CGI protocol facilitating program execution on a Web server, Web sites could now provide the means to display dynamically changing content to Internet users.

Shortly after CGI was adopted, Web site authors were creating programs using Perl to support more complex features online. It was now possible to automate Web site features that changed very often, such as news reports, stock quotes, and sports scores, all of which previously required a human to sit at a computer and update files on a server.

Even more enhanced real-time features were developed, such as user authentication, data validation, and database access. This allowed Web sites to produce larger portals that were designed to only allow registered users access to secure information.

Introducing the Common Gateway Interface



The Common Gateway Interface, or CGI, is a protocol used by the Web server to communicate with other programs that are stored locally on the Web server. These programs use CGI to identify unique information about the user's session. When a user directs her Web browser to your Web site, you can instruct the Web server's CGI *handler* to execute a custom program, like a Perl script, to generate dynamic HTML code, and relay it back to the user's browser.

The CGI acronym has many meanings. More commonly it refers to the Computer Generated Imagery created for television and movies. However, in the context of the Internet Web browser, server, and this book, CGI only refers to this communication protocol.

The output of a typical CGI program is most often HTML code, but it can be of any file type. See the section, "Understanding CGI from the Web Browser's Point of View," for a description of how this works.

CGI Process Flow

When a user requests a Web page, the Web server decides whether or not to launch a CGI process based upon the filename the user requests. If he requests a static file, such as `index.html` or `report.txt`, the server relays the file back to the user as-is. If he requests a dynamic Perl script, such as `index.pl` or `report.pl`, the server launches the CGI process, executes the Perl interpreter, and relays the program's output back to the user.

HTTP Headers

Regardless of what the user is requesting or receiving, all communication traffic handled by the Web server is prefixed by a series of *HTTP headers*. These headers supply information that is vital for CGI, but they are only visible to the Web browser, server, and CGI programming languages like Perl. The user never sees these headers when the browser renders the Web page.

HTTP Request Headers

The Web browser sends HTTP request headers to the Web server on every page request. This tells the server which URL is being requested, what languages and encryption protocols are supported, any established cookies, and any submitted content from the user through an HTML form.

CGI scripts have access to these request headers through the server's environment variables. Perl CGI scripts interpret environment variables by way of a special variable called `%ENV`.

HTTP Response Headers

The Web server sends HTTP response headers back to the Web browser, followed by the requested content. This informs the browser on how much data content to expect, how it is formatted, and any new cookies that the browser must store.

CGI scripts have limited control of the HTTP response headers. To conform to HTTP standards, the Web server provides the majority of response headers automatically but still expects the CGI script to provide the *content-type* HTTP response header. This allows the CGI script to forewarn the Web browser, advising whether the output data it is sending is HTML code, a JPEG image, or a downloadable Zip file.

HTML Forms

In order to collect information from the user, data is submitted through an HTML form. You can use this form to collect any type of information, and bind the user's typed answer to a specific identifier. When the user clicks the Submit button, a new URL is requested and the data is relayed to the server.

On the server-side, you need to configure a CGI script to collect the information being submitted and process it. You must configure the HTML form to use a specific method to encode and relay its information to the server. The two most popular methods are GET and POST.

GET Method

The GET method appends the submitted content onto the end of the URL receiving the request. Due to the limited length of URLs in some browsers, which only support 256 characters, not much data can be submitted with GET. Also, the actual data is visible in the URL's Address bar when the new page loads.

POST Method

The POST method sends the submitted content immediately after the HTTP request headers. This allows for more data to be sent than GET, and it is hidden from the browser's Address bar. The main difference of the POST method is that bookmarks and page reloads may not work as the user expects; for example, the actual submitted information might not be maintained if the user refreshes the bookmarked URL at a later date.

Common Environment Variables

Several environment variables are available to you that are passed in the context of CGI. The HTTP request headers and the Web browser populate most of these variables, but the Web server also provides some of them. You can use them within a Perl CGI script to control the interaction with the user within the CGI protocol.

VARIABLE NAME	DESCRIPTION
SERVER_NAME	The server name responding, according to the browser
REQUEST_METHOD	The method of the request (GET or POST)
HTTP_USER_AGENT	A string identifying the user's browser
HTTP_COOKIE	A string of active cookies relative to the user's session
QUERY_STRING	Any additional data passed after the question-mark (?) in the URL for GET requests
SCRIPT_NAME	The script being executed, from the perspective of the server
REQUEST_URI	The script being executed, from the perspective of the browser
REQUEST_ADDR	The user's IP address that originated the request
SERVER_ADDR	The server's IP address that received the request

Additional environment variables may be available; this depends on the type of the request, the content, the Web server, the Web browser, and any other technologies such as secure sockets layer (SSL) encryption, or server-side includes (SSI).

Understanding CGI from the End-User's Point of View

When a user directs her Web browser to a particular page, she may never know if she is requesting a dynamic CGI program or static file, or have any idea about the CGI programming language being used on the server. Even if she does recognize that the page is dynamically generated, it is nearly impossible to identify what program is being used behind the CGI interface on the Web server.

Any astute user who has an idea about CGI development and Web servers may identify several clues from a Web site, thus identifying what software it runs. Even as you

start generating CGI pages, you may start to notice some of these subtle clues on other Web sites and infer what they use to generate and display their content and services.

The problem with these “subtle clues” is that malicious users may be able to take advantage of your Web site, after identifying what software it runs, and leverage any number of known security attacks against your server.

Ultimately, these types of attacks are fairly easy to circumvent. See Chapter 22 for simple tips on preventing common attacks.

The HTML Interface

When most users visit your Web site, they only experience and interact with it through their Web browser's window. By default, this is all most users can see; however, users who are interested in how your Web site is constructed can still access the actual HTML code and syntax used.

Accessing the HTML Source Code

The Web site's HTML code is the only source-level content the user can access directly. Most Web browsers support an option to view the page's source code by simply right-clicking on the page and selecting View Source.

There is nothing you can do to prevent this. Regardless of whether a page is static or dynamic, the user can always view the HTML content.

Naturally, if a CGI program generated this content, there should be no indication of what program you used, or the contents of the CGI program's source code.

Accessing the CGI Source Code

When you properly configure a CGI protocol on the Web server, any requests by the user to access a CGI program by its URL are met with the program's output, not the program's file content.

In the case of a Perl script, a user may access the URL as `http://servername/cgi-bin/search.pl`. The CGI handler knows that any files in the `cgi-bin` directory should be executed, and not read. If your Perl script exists in another directory, one that does not have the CGI handler enabled, then the full source is visible.

For this reason, you need to properly secure your server hardware, and ensure that only appropriate people have access to the Web site source code on the server. In the previous example, if someone were to carelessly copy `search.pl` into another directory that lacks a CGI handler, all sensitive data within would be exposed online.

Prompting for User-Submitted Data

A CGI program needs to be developed that accepts any user-submitted data provided by an HTML form. When the browser first displays the form to the user, the form defines which URL should receive the data, and the method to use. It is that URL that the browser goes to when the users clicks the Submit button. The CGI program responding to that URL must collect the data, process it, and display an appropriate message back to the user.

Building an HTML Form

Most HTML forms follow the same structure. A form element introduces the form and surrounds several input elements, which collect data in the browser. The following is just an introduction to basic HTML forms. For more information, consult the W3C HTML specification.

Start a New Form

An HTML form always begins with `<form method=method action=url>`. The `method` is either GET or POST, and the `action url` should be a CGI script that opens when the user submits the form.

Single-Line Text Input

You can use the HTML element `<input type=text name=text>` to create a single-line text input field. The `name` attribute is used later by the CGI script as a key to the value provided by the user. You can use an additional attribute, `value=text`, to pre-populate the form with a default value in the field.

Multi-Line Text Input

To handle multiple lines of input from the user, use `<textarea name=text></textarea>`. The `name` attribute works the same way as the single-line input, but any default text should instead be defined between the opening and closing `textarea` elements. Additional attributes such as `rows=num` and `cols=num` can control the dimensions of the multi-line text input box.

Hidden Text Input

Use the HTML tag `<input type=hidden name=text value=text>` to pass additional information in the HTML form, but not allow the user to see it or change it directly.

This can be useful if one CGI program originally generated the HTML form, and it needs to send additional data to another CGI program that will process the form.

Submit Buttons

All forms should have some sort of Submit button. This is what the user clicks to send the form to the Web server. You can often use `<input type=submit>`, but a button with the literal text, “Submit Query,” displays. To change the button text, add the attribute `value=text`.

It may look weird in that this is the only input element that may have a `value` attribute, but not a `name` attribute. If you include a `name`, then the CGI program will receive the button's value as an input field. This can be particularly useful if you have multiple buttons on the same form, each with different processing functionality.

Finish the Form

Complete the HTML form with a closing `</form>` tag. This instructs the browser that the form is complete.

Viewing Data Returned by the Server

Once the data has been populated by the user, and submitted to a CGI program, the program should display something useful or intelligent to the user to indicate the results of their request. This may be as simple as displaying a “Thank You” message, maybe even stating that all information was received correctly. Or, if the user failed to correctly populate a field, the CGI program needs to inform the user which field was incorrect and why, and re-display the form. It is always good etiquette to have your CGI program pre-populate the fields that were correctly submitted with the original value, and to highlight the fields that were incomplete or incorrect.

Understanding CGI from the Web Browser's Point of View



The Web browser actually receives a lot more information from the Web server than it displays. The user is completely unaware of the subtle interactions between the browser and server, including

various CGI communications. As a dynamic Web site author, you need to be aware of what is going on here so that you can better interact with your user's browser and provide a dynamic Web site experience.

Environment Variables

All CGI programs have access to several environment variables related to each Web page request. The Web server provides some of these variables, while the Web browser provides others.

Apache comes with a useful CGI script called `printenv.pl`, which you can use to see all environment variables that are in use, and to validate that CGI program execution is working correctly. To enable this script, you must first enable the CGI handler in Apache; see Chapter 10 for more information.

The Web browser provides the CGI with several environment variables including the following: the current URL (`HTTP_REQUEST_URI`), the referring URL (`HTTP_REFERER`), the browser's language (`HTTP_ACCEPT_LANGUAGE`), and the browser's software (`HTTP_USER_AGENT`). By reading these environment variables, your CGI program knows what URL the user is requesting, what URL he is coming from, what languages are supported, and the browsing software version, respectively.

Cookies

Cookies are portions of data that a Web server or CGI program can assign to a user's Web browser, allowing it to "remember" a user from an earlier Web site visit. When a CGI program wants to remember a user, it sends an initial cookie to that user's browser using the HTTP response headers. This could include Web site preference information such as the user's preferred language, or a uniquely generated authentication token.

The Web browser's job is to accept and store the newly assigned cookie. The Web browser does not care about the cookie's content, only that it must relay that same cookie back to the Web server on any subsequent Web page request using the HTTP request headers.

If that same CGI program receives its cookie back again, it knows that this user has visited before, and retrieves the data that it asked the browser to store. It is possible for a user to configure her Web browser to reject new cookies, or manually delete existing cookies. Doing so makes it impossible for the CGI program to remember the user, forcing it to treat the user like a first-time visitor.

MIME Types

All content delivered by Web servers is preceded with a special Multipurpose Internet Mail Extension (MIME) type header. This introduces the HTTP content coming from the Web server to the user's browser. As the name implies, MIME originated in the realm of e-mail, allowing for multiple message content blocks to be bundled into a single e-mail message. This allows the client to choose how to display the content by announcing its content type. The HTTP protocol uses a subset of MIME and requires all content delivered by a Web server to specify an appropriate *content-type* HTTP response header called an *Internet Media Type*.

INTERNET MEDIA TYPE / MIME TYPE	DESCRIPTION	COMMON FILE EXTENSIONS
text/plain	A plain-text file with no special formatting	.txt
text/html	An HTML-formatted Web page	.html .htm
image/jpeg	An image saved in the JPEG format	.jpeg .jpg
application/zip	A compressed ZIP archive	.zip

MIME Types *(continued)*

At first glance, MIME types in HTTP headers seem redundant; the Web browser should be able to identify the file type based upon its extension. When the user requests the address `http://mysite.com/index.html`, she is obviously viewing an HTML file as the Web page. However, if the user requests a CGI address such as `http://mysite.com/index.pl`, the Web browser does not know what format `.pl` files represent, and neither does the Web server. Instead, the Web server blindly executes the program, relaying all output back to the browser. The CGI program provides the content-type response header in its output, and the Web server relays this to the browser. So, if the CGI program announces content-type `text/html`, then the browser knows to render the CGI's output as an HTML page.

Processing User-Submitted Data

When an HTML form displays on the browser, the user is prompted to populate its fields and click a button to submit the data to the server. Each field has an identifier that is used as a lookup to match the field's value. The Web browser's role is to collect the information from the fields and encode them in a way that can be transmitted safely to the Web server.

Encoding the HTML Form's Values

The data being sent is encapsulated within a format that uses special characters to separate fields and values. If the user happens to type in a character that is sensitive to this format, a macro must replace that character so that it does not interfere with the expected formatting.

Fortunately, all alphanumeric characters are generally safe as-is, but some non-alphanumeric characters must be converted into their *percent-hexadecimal-ASCII* format by the Web browser. For example, the equals sign (=) is represented in ASCII as value 61, or in hexadecimal as 3D. If a user types an equals sign into an HTML form, the browser encodes the character as %3D.

Some of the characters that require conversion include the equals sign, plus sign (%2B), carriage return (%0D), line feed (%0A), question mark (%3F), and ampersand (%26).

The only exception to this rule is the space character. While %20 is perfectly legal, Web browsers often convert it into a literal plus sign (+).

Receiving Data from the Web Server

The Web server never handles form data directly. Instead, a CGI program is responsible for processing the data. The program simply reverses the process performed by the Web

browser: it splits the `name=value` pairs by the ampersand, and then decodes all of the *percent-hexadecimal-ASCII* values back into their original character values.

Sending Data to the Web Server

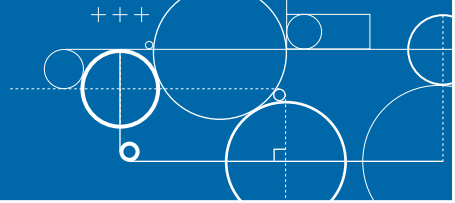
Before the data is sent to the Web server, the Web browser serializes the HTML form's fields into a string of `name=value` pairs, joining each pair with an ampersand (&). Because each `value` has had any sensitive non-alphanumeric characters encoded, these characters cannot affect the overlying structure of each pair, and the user's original value is maintained.

FORM FIELD NAME	USER VALUE	ENCODED STRING
name	John Smith	name=John+Smith&
age	40	age=40&children=
children	Chris & Jason	Chris+%26+Jason

If the HTML form specifies the GET method to send the data, the URL is appended with a question mark (?) followed by the encoded string of names and values.

If the HTML form specifies the POST method, the URL is not appended. Instead, the encoded string is sent after the standard HTTP request headers.

Understanding CGI from the Web Server's Point of View



The Web server's role is to facilitate the flow of information between the Web browser and the online content hosted by the server. This may require the Web server to execute a CGI program to provide the dynamic features of a Web site. Fortunately, because the CGI protocol has been standard for years, it does not matter which Web server or Web browser is used.

This section, in fact this entire book, uses Apache as the example Web server. If you compare it to another Web

server such as Microsoft Internet Information Server (IIS), you will see that the majority of the concepts revolving around CGI are basically the same. The only real difference is the implementation and configuration of the actual Web server.

The Web server's role is basically transparent to the Web browser and CGI program. Its sole purpose is to forward request information being sent from the browser to the CGI program, and to relay response information from the CGI program back to the browser.

Receiving Data from the Browser

When the user goes to a Web site, his browser generates an HTTP GET or POST request and sends it to the site's Web server. This request consists of the specific site URL and the user's current HTTP environment settings, along with any cookies, secure socket layer (SSL) encryption status, and any other relevant information.

This information is bundled up into an HTTP request message and sent over the Internet. The Web server receives the HTTP request, reads the information that it deems relevant, and identifies if either a static file or a CGI program is needed to

complete the request. So, if the user requests a static HTML file, then a CGI program does not run. The contents of the static file are relayed back to the user, encapsulated in an HTTP response message.

If the user requests a file that identifies itself as a CGI program, then Apache recognizes this, locates the program on the Web server's hard drive, and executes it appropriately. The program's output is sent back to the Web server, which it bundles with a similar HTTP response message.

Executing a CGI Program

A CGI program can be either a compiled binary or a standalone script. The Web server needs to know how to execute the program so that the intended output is sent back to the user.

In order for the Web server to identify if a CGI program is required, the server must have an understanding on the various ways a CGI program can be called. For example, it could be called directly, by the user specifying a Perl script in the Web browser's URL, or indirectly, hidden within an SHTML file using *server-side includes*. Once the CGI program is identified and located, it is executed.

Executing a CGI Program *(continued)***Forwarding Data to the CGI Program**

The Web server relays the CGI data to the CGI program in multiple ways. Regardless of the HTTP method used, the program's environment settings are populated with CGI data. Also, if the HTTP POST method is used, additional information can be found on standard-input.

It is the CGI program's responsibility to interpret, parse, and analyze the CGI data into usable segments. In the case of a Perl CGI script, it should relay this information to the programmer in a way that is easily accessible.

The information being sent to the CGI program includes any request data, cookies, HTML form fields and values, the user's environment settings, the Web server's environment settings, and any other information relevant to the request.

Receiving Data from the CGI Program

Once the CGI program is launched and performs its function, it sends its generated output back to the Web server by way of standard-output. In other words, the CGI program simply *prints* the output back to the Web server.

Prior to sending any of the actual data, though, the first line of the CGI program's output must define a content-type MIME header. Because a CGI program can technically output any type of content, be it an HTML-formatted Web page, a JPEG image, or a Zip file, it must introduce the content type to the Web browser. This is required because the CGI program cannot output the extension to the actual file format, such as HTML, JPEG, or XML. The extension simply does not exist in the HTTP response message.

By providing the content-type, the Web server knows the formatting of the CGI program's output, and can relay this information back to the Web browser, which then handles it appropriately given the content. Ultimately, the Web server does not care about the format of the data it receives from the CGI program. Its sole purpose is to send that data back to the user's Web browser.

Sending Data to the End User

After the initial request for the Web page, the user sits and waits for his Web browser to render the page. The Web browser waits for the Web server to return the results of the request. The Web server in turn waits for the CGI program to return its output. All this time, the CGI program may be using the server's CPU processing power, calculating some complex task. Only when the CGI program finishes does the process unravel: the program's output is sent to the Web server, then to the Web browser, and then to the user.

As data flows back to the user, it changes slightly with each step. The Web server actually appends additional HTTP response header fields to what it receives from the CGI program, completing the HTTP response message.

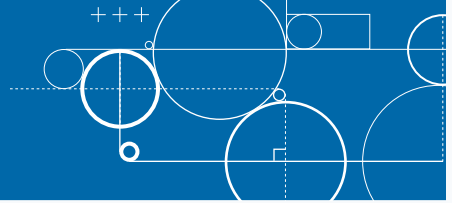
Naturally, problems can happen in this process. Most often, it may be due to a problem in the actual CGI

program's execution. If the program crashes, then no output is sent back to Apache, which in turn sends a generic error message back to the user, something like "Error 500: Internal server error."

Or, if the CGI program takes too long to respond, Apache simply gives up waiting. Instead, it sends a timeout error back to the user, perhaps with some bundled text asking the user to try again later.

Regardless of the response from the CGI program, Apache needs to send something back to the user's Web browser as its HTTP response message. Apache constructs the response data, based upon conditions set out by the request. It tacks on the data received from the CGI program if applicable, or the contents of the file requested, sending everything back to the requesting user's Web browser over the Internet.

Understanding CGI from the CGI Program's Point of View



A Web site's author may choose to build her site with HTML content split into several individual files, such as `index.html` or `aboutus.html`. Static files are relatively easy to construct, but makes it difficult to provide any type of dynamic content. Instead, the author may create one or more CGI programs to dynamically generate each Web page.

The CGI approach has the advantage of controlling what information displays, and making it unique to the end-user given his particular session. This may be as simple as including the current date and time in the top-right

corner of the Web site, or as complex as an e-commerce store with a shopping cart and a credit-card processing checkout.

The CGI handler provides the conduit between a CGI program, such as a Perl script, and the user. The program must collect the incoming information provided by the Web server through the CGI handler including the user's environment, cookies, and session. The program must analyze this information, process it, and generate output from it, like a dynamic Web page.

CGI Handler Executes the Perl Interpreter

For every user who requests a Web page that is dynamically generated, the Apache CGI handler must launch a new instance of the Perl interpreter into memory to process the request. The CGI handler also instructs the Perl interpreter which Perl script it needs to execute, and finally forwards the incoming CGI data to the script's run-time input.

Naturally, if the user references a Perl script filename in the URL, Apache does not necessarily know that it is executing a Perl script, or a binary file written in machine language. Because a script file is nothing more than text that follows a specific syntax, it cannot be executed directly by the operating system like a compiled binary.

In the Unix world, when a program file is launched on the command line, the operating system relies the *system shell* to identify if the file is binary and can be executed directly, or if it requires a helper program. To do this, the shell reads the first line of the file, looking for a *shebang* (`#!`) interpreter directive. This is why all Perl scripts begin with the following line:

```
#!/usr/bin/perl
```

The shell sees this and knows that it must execute `/usr/bin/perl` first, as this is the program that can properly interpret the contents of the script file.

In the Windows world, the same basic thing happens, except the `explorer.exe` "shell" does not look for an interpreter directive. Instead, it recognizes Perl scripts by way of a `.pl` extension. It knows that `.pl` files first require the program `C:\Perl\bin\perl.exe`. When executing CGI scripts on Windows, Apache does not use `explorer.exe`, but instead assumes the role of a Unix system shell. Apache opens the file being referenced and looks for a shebang interpreter directive, which must describe the location of the Perl interpreter binary installed on Windows:

```
#!C:/Perl/bin/perl.exe
```

Reading Data Sent from the CGI Handler

Once the Perl interpreter is running, and it has parsed the Perl script and executed it, the first thing the script should do is read introductory information about the HTTP request session from the CGI handler.

The CGI handler supplies most of its information to Perl by way of environment variables into the Perl interpreter session. The CGI handler provides the URL requested, the user's IP address, the Web browser name and version, all cookies related to the Web site, and so on. All of this information is gleaned from the Perl session's global environment variable.

Sending Data Back to the CGI Handler

It is the Perl script's job to enhance the user's Web-browsing experience. Usually this happens by way of dynamically generated HTML, but it could be any type of formatted data, composed by the Perl script's logic. The Perl script simply "prints" this information on its standard-output handle, with Apache listening intently to this handle while the Perl script is running.

However, prior to printing any HTML code, the Perl script must first print any outgoing HTTP headers. Apache relays these HTTP headers back to the user's Web browser in the HTTP response message. In essence, these headers allow

the Perl script to communicate directly to the user's browser, within the guidelines of the HTTP specifications.

At a minimum, the Perl script must print the content-type MIME header. This is absolutely required, and enforced by Apache. If it is missing, Apache sends a generic error code back to the user, regardless of the Perl script's output.

Additional headers may also be provided to further fine-tune the HTTP session. Cookies, for example, are also created by way of the outgoing HTTP headers. After the Perl script defines any outgoing headers, it must print one blank line, followed by the actual generated content.

Shutting Down

Once the Perl script has finished, and its output has been sent back to Apache for relaying to the user's Web browser, the Perl interpreter closes its connection to the CGI handler and shuts down.

Normally, Apache waits until the Perl interpreter has exited before sending *any* data back to the user. Apache actually buffers the Perl script's output and only sends it once the CGI program is complete. Most of the time this is not an issue; however, if you have a Perl script that takes several seconds, or even minutes, to run, the user will be waiting for some time before the browser displays the results of the Web page request. If the delay is unavoidable, you can at least instruct Apache to provide *some* content back to the user, such as a "Please wait..." message, by flushing the output buffer in the middle of program execution.

Given enough traffic on a dynamic Web site, constantly starting up and shutting down the same group of CGI programs can be rather demanding on the CPU. In the case of a Perl script, the Perl interpreter is the actual binary program that is being launched on each request. For every single request, the interpreter parses the script into machine code, executes it, and exits. It is possible to configure Apache to actually embed Perl directly into itself. This means that the Perl interpreter is persistently running in RAM within every process of Apache, so you can avoid the actual startup and shutdown of Perl; the interpreter simply remains idle until a CGI request is received. This feature is provided by an extension module for Apache called *mod_perl*. See Chapter 23 for more information on implementing *mod_perl* on your Web site.

Compare Perl to Other CGI Languages



Although this book focuses on using Perl as a CGI language, Perl is certainly not your only option. Technically speaking, any programming language that can access environment variables, read incoming data, and write outgoing data will support the Apache CGI handler.

Not all programming languages are created equal. Some have libraries specially designed to access CGI data, while others have modules that make constructing HTML very

easy. Some have a language syntax that is very easy to remember, and others have third-party modules that make complex calculations very simple.

You may find that as you experiment with different languages, some work better than others. There is no rule that says you must select one specific language for an entire Web site. Feel free to mix and match languages and technologies and see for yourself what works best.

PHP

PHP is an interpretive language that was designed to specialize in CGI development. While influenced by other languages such as C, JavaScript, and even Perl, PHP makes building dynamic Web sites very easy.

PHP's strength comes mainly from its ability to embed itself directly within an HTML file. Static Web site content is represented as standard HTML in a PHP script. Content that is dynamic is written as PHP code, contained within special `<? ... ?>` tags.

A lot of the mundane CGI handler interactions are automated by PHP. For example, the PHP interpreter takes care of tasks

such as parsing HTML forms for values, and accessing details from the CGI environment. At a minimum, Perl requires you to either import its CGI library, or to produce 15 to 20 lines of code by hand, to accomplish the same thing.

PHP does have its flaws. For example, its ability to parse, sort, and organize raw data is not as efficient as that of Perl. Also, its library of third-party modules is not as robust or mature as the Perl CPAN repository. For simple CGI programming, however, many developers find PHP to be more than adequate.

Active Server Pages

Active Server Pages, or ASP, is a Microsoft framework for dynamically executing server-side code in-between standard HTML. First released in 1996, ASP refers to the engine on the Web server; the underlying language is actually VBScript.

At first glance, ASP-formatted files look like PHP files, using the tags `<% ... %>` instead of PHP's `<? ... ?>`. However, the feature set available to VBScript is much more robust than PHP, especially with the ASP engine providing a series of object handles that you can use to manage the application, as well as request, response, server, and session classes.

The VBScript language is limited to running on Microsoft Windows-based hosts; however, it is not limited to IIS. A third-party module is available for making ASP and VBScript Web sites run under Apache on Windows.

VBScript supports many standard functions found in most programming languages, such as built-in file, date, math, string, and binary methods. ActiveX objects provide additional functionality, but design problems in ActiveX and VBScript have led to problems with malicious third-party code, and several security advisories and patches.

ASP.NET replaced ASP in 2002. The shift to ASP.NET meant that it is no longer an interpretive language; code must now be compiled natively within the .NET framework. While this does mean more overhead in terms of memory utilization, the code is generally faster and more responsive than the original ASP. Many ASP.NET developers have chosen to continue using *classic* ASP, especially to develop simple Web sites.

Ruby on Rails

Ruby on Rails is an application framework for developing dynamic Web sites using the Ruby programming language. Ruby is an interpretive language that has been around for years, and is influenced by other languages such as Perl and Python. However, Ruby on Rails is a fairly recent framework that is gaining in popularity. It makes key assumptions about common features and functionality, and strives to maximize code re-use.

Directly comparing Ruby on Rails to Perl is not exactly fair. Ruby on Rails enforces a specific architecture method to

group the Web site development into interaction, display, and structure components. By forcing the programmer to separate her code into these three components, Ruby on Rails developers argue that Web sites can be built more efficiently and quickly. Perl CGI does not mandate that you design using this model, but you are welcome to if you prefer this type of organization.

Some developers of larger Web sites have criticized Ruby on Rails for not scaling efficiently, even opting to mix technologies such as Perl on more resource-intensive operations.

C, C++, C#, and .NET

Programs written in C, C++, C#, or .NET are compiled into binary-code, which is executed natively by the operating system. With these languages, there is no interpretive program involved. Instead, these programs need to be compiled once, and then executed by the Web server each time it receives a CGI request.

This method of developing CGI programs is very different from the other examples in this section; however, natively compiled programs do have their own unique strengths and weaknesses that you should consider.

One huge advantage to using one of these languages is that the program is already converted into a format that the CPU can quickly and efficiently execute. This means that larger programs can run much faster and are generally more responsive than interpretive-based languages.

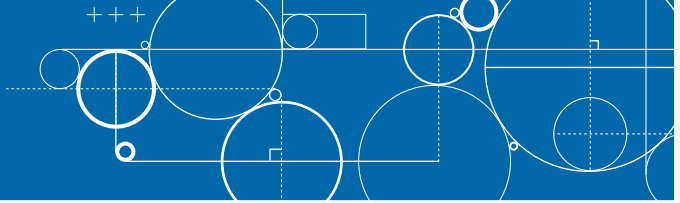
Strictly speaking, using an interpreter means that your source code is re-compiled for every single CGI request that the Web server receives. In addition, the actual execution of the interpreter affects the server's memory and CPU resources.

The disadvantage to using a native language is that there is more work involved in creating and debugging a program, as compared to a similar program that you would write using an interpretive language. It is more tedious because the programmer must manually compile the binary object before installing it in a location where the Web server can access it. With an interpretive language, the compile-and-install steps are not applicable.

The programmer must also worry about lower-level memory management, and system-level hardware interaction — something that is taken care of automatically by a higher-level interpretive language such as Perl or PHP.

Ultimately, for established Web sites that are larger, busier, and make significant resource demands on server hardware, a natively compiled binary is a better long-term solution, provided that you can invest the extra effort into writing and maintaining the program. For a newer Web site with little traffic or resource demands, using an interpretive language such as Perl will help you develop the site more quickly.

Compare Apache to Other Web Servers



Just as many other languages have implementations of the CGI protocol, so do many Web servers. As of the March 2010 Netcraft survey of Web server software, Apache represented a 54 percent market share of all Web sites surveyed — over 112 million servers. Naturally, the software must be good if so many people are using it; however, some find it overly complicated and difficult to set up, especially with its text-based configuration files. Programs such as Microsoft Internet Information Server (IIS) have stolen market share from Apache partly because they are easier to configure and deploy.

Unless you run your own server hardware on the Internet, you may not have much of a choice regarding which Web

server to run on your Web-hosting provider's servers. Apache is the standard on practically all Unix-based hosting providers. Windows-based hosting providers tend to only provide IIS as an option. However, it is strongly recommended that you install a Web server locally on your workstation for development and testing purposes.

When choosing a Web server, at a minimum you need something that supports CGI. Additional support for features such as server-side includes, secure socket layer encryption, virtual domain hosting, and basic/digest authentication are all certainly nice to have, but your specific application may not require them.

Internet Information Server

Microsoft first released IIS 1.0 in 1995 with Windows NT 3.51. Since then, the program has been steadily upgraded and improved with each new release of Windows. The most recent major version is IIS 7.5 for Windows Server 2008.

As of March 2010, about 24 percent, or roughly 50 million Web sites, use IIS. This is impressive because only Windows servers can deploy the program; however, IIS's level of adoption has bounced between 20 percent and 40 percent since late 1997. Since peaking in late 2007, its market share has dropped to a new four-year low.

IIS and Apache have been competing for market share for many years. In fact, the two programs have become so synonymous with their respective operating systems that many Web site developers have simply accepted either program as the default option — immediately after selecting a development platform. The key advantage Apache has over IIS, with respect to market share, is that Apache can run on either Windows or Unix servers, whereas IIS is limited to Windows-server platforms only.

Unlike Apache, IIS's license and source code are proprietary, and not freely available. Rather than charging for it, Microsoft

has been shipping it out as a standard component of all Windows installs since Windows XP.

Strictly speaking, as a CGI Web server, IIS is fully compatible with Apache in terms of handling Perl or other interpretive languages. In other words, if you use IIS on your personal workstation to develop and test your Web site, and Apache on your Web-hosting provider's server to deploy it publicly, your CGI scripts should be equally usable at both sites, regardless of any minor compatibility adjustments to the CGI source code.

Depending on whom you ask, several studies have been released that argue that Apache is better than IIS and vice versa when compared on similar hardware. The only real disadvantage to IIS may be its performance serving CGI scripts under high-volume conditions; after all, IIS is designed to run ASP, a competitor to the CGI standard. You can use a third-party program called FastCGI to address this CGI deficiency on IIS. FastCGI runs underneath IIS and acts as sub-server for all CGI traffic deferred by IIS. FastCGI is also available for Apache.

nginx

The Web server program nginx, pronounced “engine-X,” was the surprise contender on the March 2010 Netcraft survey. More than 13 million Web sites, a 6.74-percent share, were counted as running nginx, a surprise in that this program was only first registered on the survey in early 2008, but also because it has just one developer.

Nginx supports most of the basic features that are standard in the popular Web servers; what makes it special is that whatever feature it does not support can be added using a well-documented plug-in API. One feature sorely lacking in nginx is native CGI support; fortunately, it is compatible with a FastCGI background process where, when enabled, nginx acts as a proxy to FastCGI.

The real strength of nginx comes in load balancing, fault tolerance, and serving static files. For this reason, it is not the best choice for CGI-centric content. Instead, it is best suited for busier Web sites that want to split the CPU load of static and dynamic content from a single server farm. You can let nginx handle the mundane tasks of serving the static HTML and images, and let the more powerful machines running Apache handle the dynamic work.

On its home page, <http://wiki.nginx.org>, you can read about downloading, installing, and configuring nginx on Unix, Windows, and Mac platforms.

lighttpd

Lighttpd is advertised as a lightweight, fully featured Web server that focuses on being secure, fast, and flexible. As of March 2010, its share was reported to be 0.53 percent of all domains, or slightly over one million. As its level of market share grows, it boasts that it can comfortably handle more than one thousand hits per second on a single server managing static Web content.

The default installation is fully CGI-compliant. You can further improve the level of CGI responsiveness by adding the aforementioned FastCGI program underneath the lighttpd public serving layer.

Similar to Apache, lighttpd’s configuration uses files; however, the syntax is more akin to programming statements such as `config key = value`. It is very

impressive that a Web server such as lighttpd is very small and yet of such high quality. The source code and installation footprint of the core server is roughly 10 percent the size of a comparable Apache footprint. The only real problem with lighttpd is its small developer and support base — only 12 people compared to hundreds for Apache.

The lighttpd source code is available at www.lighttpd.net with pre-built packages available for various Linux distributions. Using lighttpd under Windows is possible, but you need to compile and run the code within a Cygwin environment. Cygwin is a separate project that has ported many useful Unix-based libraries and utilities into the Windows operating system.

Developing Your Web Site

When first starting out as a developer, learning to program in a new language involves an investment of time and an expectation of problems. The degree of which varies from programmer to programmer.

You will also find that the various third-party programs, techniques, and recommendations discussed in this book or on the Internet actually contradict what you know works for you. By keeping an open mind, experimenting with new ideas, and building upon what you have learned, you become a more proficient programmer over time.

Your Development Environment

The development environment refers to your local workstation, programs, and documentation. This is where you will be spending your time actually developing the code for your Web site. You will need several tools and techniques in order to efficiently create a new program. Feel free to experiment with what works best.

The Vim Text Editor

The text editor is the primary tool that you use to write code, and this is where you will be spending most of your time programming. Choosing a text editor involves experimenting with the different editors that are available. One extremely powerful editor is called Vim.

Vim is a console-based text editor that allows you to use colored syntax highlighting, automatic code spacing, command completion, and macros; it is also extendible through hundreds of plug-ins. One common complaint with Vim is that its method for entering commands is very different from other editors. The commands will seem too imposing and complex, and many new users will find concentrating on them gets in the way of concentrating on the code. However, if you simply learn five basic commands (open, save, insert, move, and quit), Vim will start to pay dividends in productivity.

A simple tutorial designed for people starting out is standard with every installation of Vim. Just run the command-line program, `vimtutor`, which is available for both Windows and Unix installations.

Coding Style

There are many guidelines available on the Internet that summarize the best coding style to use. Perl is very open in its interpretation of your code; however, some argue that it is “too open.” This may lead you to develop code that is messy, obfuscated, and difficult to maintain.

Ideally, you need to maintain readability with your code. This includes using appropriate variable names, function names, and spacing.

One standard that is widely accepted by the open-source community is the GNU Coding Standard. This document describes everything in a C-based model, but there are sufficient similarities with syntax that it translates easily to Perl. The document is available in HTML and PDF format at www.gnu.org/prep/standards/.

Comments

Commenting your code is never a bad idea, but there is no need to write comments inefficiently and excessively. With a little extra effort, you can create comments that are concise and effective. A good rule to follow is this: if it is not inherently obvious what a portion of code does, precede the code with a little comment that explains it.

Comments are also used to introduce subroutines. They explain the subroutine's purpose, what variables it accepts as input, and what it produces as output.

Regarding more complex code, if you have to spend several minutes building the logic for a specific piece of functionality, then comment your development thoughts and efforts like a mini-diary. You will save yourself time if you ever need to revisit that particular segment of code.

Project Deployment

Regardless of the size of your Web site, a good project deployment policy means having distinct server environments set up for your site. Each environment focuses on a specific state of your code: new work occurs on the *development environment*; it is validated on the *testing environment*, and finally deployed to the public on the *production environment*.

Controlling the flow of new code in these three stages ensures that your experimental development environment stays private, and your public production environment remains stable.

The Development Environment

The purpose of the development environment is to house newer, experimental code. This includes changes to your Web site that could either perform correctly or crash. The point is that you do not know until you place the change somewhere on a Web server and try it for yourself.

This environment should remain hidden from the general public; only internal developers should have knowledge of it. This environment may exist directly on your individual workstation, or be managed by a dedicated development server.

You will find that your production environment matures with real-world data over time. Being able to import this data into your development environment can be very helpful. This is always a good idea just prior to starting on a new development task, as you will gain a unique perspective on how users are actually interacting with your Web site.

The Web server software that runs on the development environment should be either the same version or newer than the test and production environments. This is required in order to validate whether a potential server upgrade will have an adverse effect on the actual Web site.

The Test Environment

The test environment is intended for relatively stable code, and is used to find any remaining bugs and do final quality assurance testing. Because the developer promoted all of

the code here from his development environment, no code is expected to crash or fail.

This environment should be a dedicated test server, separate from the development and production servers. With respect to server hardware, it should be as close to the production environment as physically possible. The primary people using this environment should be testers in charge of quality assurance. They must validate specific bugs have been fixed, new features work, and that there are no regression problems.

The URL for accessing this environment should be restricted to internal testers; however, you may find it beneficial to provide regular users access, and try to gauge their opinion. From here, you decide if the change is safe to be promoted to the production environment, or demoted back to development to be fixed.

The Production Environment

The production environment is the public's window to your project. This is where all Internet traffic goes, so obviously this has to be the most stable code available, with no known bugs or issues. Everything must work as designed.

Only code that passes quality assurance testing can be promoted from the test environment. Absolutely no new code can just suddenly appear here without first going through development, and then testing stages. Think of installing new code into the production environment as a quasi "software upgrade," complete with a new version number and change log.

Find Perl- and Apache-Friendly Hosting Providers



Once your new Web site is built, you need to make it available online. An Internet *hosting provider* allows you to lease a portion of their professionally managed servers to your Web site, granting you access to Perl and Apache. Many Web developers find hiring a hosting provider to be very beneficial because they offer a dedicated high-speed Internet connection, professional staff for monitoring and

maintaining the network, and support staff in case any problems arise.

You also need to register your Internet domain name with a *domain registrar*, which may be separate from your hosting provider. A registrar reserves a specific domain name and offers you a *domain name service* (DNS) assignment program, allowing you to link your domain name to the hosting provider's network.

Compare Hosting Plans

Hosting providers offer various plans that range in price, service, bandwidth, and level of server access. Selecting a specific company's plan depends largely on your needs, budget, and anticipated Web site growth.

Simple Domain Hosting

Often the cheapest option available to Web developers, simple domain hosting implies the bare minimum required to put a static Web site online. This is not usually sufficient for sites that require Perl and Apache, nor is there any access to the server hardware. Access is usually only made available through file transfer protocol (FTP).

Some hosting providers may offer some form of CGI access with these plans. Usually this includes a default installation of the Perl interpreter, except with no extra third-party modules or enhancements, or even the opportunity to apply them yourself. This can be very limiting for a complex Perl CGI Web site, but may be sufficient for a site with only simple Perl CGI scripts.

Virtual Hosting

A virtual hosting service plan gives you access to a stand-alone Linux or Windows operating system installation that is virtually hosted within a single physical server. That single server may have multiple virtual accounts, just like yours, each one sharing a slice of the host's CPU and memory resources. Each virtual server sees itself as a complete operating system, segregated from its virtual peers and the host operating system.

Virtual hosting is best for startup Web sites as the monthly leasing costs are fairly low. Often you are granted unrestricted access to a root account if Unix-based, or the administrator account if Windows-based, by way of a remote access program such as Remote Desktop or an encrypted terminal

program called SSH (secure shell). This grants you full rights to the virtual server, allowing you to install your own third-party support programs or modules, as required by your Web site.

One disadvantage to virtual hosting is that your virtual neighbors may suddenly execute a CPU-intensive task that draws most computing resources away from your own site. This can happen sporadically, and without warning. Usually, there is no guarantee for virtual hosting providers regarding available CPU resources. This means that the actual level of computing power may not be consistently as advertised.

Cloud Hosting

A cloud-based hosting plan is very similar to a virtual hosting plan in terms of usability and execution. However, instead of a single physical host server, there could be hundreds or even thousands of clustered host servers, synchronized together, acting as a singular cloud entity.

The cluster could manage any number of virtual operating system environments at once. Each one is allocated a specific slice of the collective CPU strength of each cloud node.

Cloud hosting has the advantage of scalability and cost effectiveness. As the CPU power is evenly distributed, you are much less likely to encounter a sudden reduction in CPU resources as with a single-host virtual hosting provider. Even if you do need more processing power, you can simply lease a larger slice of the cloud service.

Compare Hosting Plans *(continued)*

Dedicated Hosting

When outsourcing your hosting services, the most expensive plan is dedicated hosting. This means that a dedicated physical machine is set up for you, and your Web site is the only resident.

Dedicated hosting is offered in two ways: by leasing server hardware or by co-locating hardware. In both cases, you

pay for the bandwidth used, which may be a flat or variable rate, but when leasing hardware you also pay for the rental of the physical machine. Some dedicated providers offer professionally managed maintenance and security services, which also add to the cost.

Dedicated hosting is best suited for established Web sites that service a lot of traffic, and where users expect a certain level of performance and responsiveness.

Use Your Own Server

Many Web site developers forgo the cost of leasing a hosting provider's server and instead choose to deploy their Web site locally, running through a home or office Internet connection. While there is nothing inherently wrong with this, you are taking responsibility for purchasing, maintaining, and securing your own server hardware. Before you begin serving incoming Internet traffic, you need to be aware of your Internet service provider's (ISP) policy on serving an outgoing connection; you may need to purchase an upgraded connection.

Your local server needs to be connected to the Internet with a *fully routable* IP address. This means that there is no router or firewall with network address translation (NAT) enabled in between your server and the Internet connection. You can tell if you have a fully routable IP by comparing your computer's locally reported IP address to a service such as www.whatismyip.com; if the values match, then your IP is fully routable. Many ISPs also enforce automatic IP address rotation to discourage self-hosting on their standard end-user network. Again,

purchasing an upgraded Internet connection ensures that you have a *static* IP address, which makes DNS assignments to your domain name significantly more stable.

You still require a firewall to control access between your server and the public Internet. Not enabling a firewall creates a very high security risk, depending on your operating system. Usually you should configure the firewall to block all incoming connections from the public Internet except for what is minimally required, such as TCP ports 80 for HTTP and 443 for HTTPS.

Many retail ISPs provide a high-speed connection that is designed only for regular end-user traffic, such as browsing Web sites, downloading videos, and playing games. If your ISP installs a NAT firewall, blocks port 80, or enforces frequent IP address rotation, anyone looking to serve their own Web site will be stymied. In this case, there is nothing you can do other than to pay for an upgraded connection, or hire a professional hosting provider.

Find Help Developing CGI Programs

As your Web site becomes more dynamic and complex, you will encounter problems and have development questions. This could mean tackling problems with your source code syntax, its execution, responsiveness, or usability. You should address these problems as quickly as possible as they could negatively impact the experience of end-users.

In the case of a Web site with many CGI programs, there can potentially be a large number of independent parts, including the Perl binary, various third-party libraries, and your actual program. Deciding where to go for help

depends on you correctly identifying the failing component. If the failure is a misunderstanding of how to use the component, you should always review its documentation first.

There are many resources available to provide you with help writing Perl CGI code, and these are available online and locally within your Perl server's documentation. Quickly locating these resources allows you to deal with the problem sooner, potentially avoiding any major site downtime and unhappy users.

Documentation

The support documentation that describes how a program works is extremely important. If there is no documentation available to diagnose a problem or describe a feature, the end-user can feel lost, disillusioned, and generally disappointed in the program's original author. Effectively, the program is now useless to the end-user if he cannot figure out how it works. Fortunately, Perl and Apache have extensive documentation libraries that are frequently updated and reviewed after every software release.

Perl Documentation

Perl includes extensive documentation that is installed by default alongside the Perl interpreter. This documentation describes every built-in Perl function, module, and command-line argument, and even provides an in-depth series of FAQs and tutorials.

Local Perl Documentation

You can access the Perl Documentation locally by way of the command-line program `perldoc`. You can access individual documentation pages by appending the applicable page name as an argument:

```
perldoc pagename
```

For example, the Perl Documentation's table of contents is available under the page name `perltoc`. You can view it to display all the other pages that are available in the Perl Documentation. The exhaustive Perl FAQs is also an excellent resource, and its summary is under the page name `perlfaq`. Installed, built-in, and third-party modules also make their documentation available by way of `perldoc`. Simply append the module's full class and name as a parameter:

```
perldoc Class::Name
```

You can access the syntax for individual Perl functions using the `-f` argument. This is a great way to review the specifics about how an individual function works:

```
perldoc -f function
```

You can find a complete list of all available Perl functions under the page name `perlfunc`.

Documentation *(continued)***Online Perl Documentation**

In case the local Perl Documentation is inaccessible or not installed, the latest version is available in HTML and PDF formats at <http://perldoc.perl.org>. Many Perl developers find it easier to access the Perl Documentation on this Web site than through the command-line `perldoc` program. This may be because the local documentation program works under the assumption that you already have a rough idea of what you are looking for, and so it excludes the table of contents page; as a result, there is no real way to simply browse through the Perl Documentation library other than through the Web site. The Web site also has the advantage of being able to cross-link specific terms from one page to another, something that is impossible to do in `perldoc`.

Ultimately, the command-line program is better suited for quick, one-time references to a specific module or function; however, when learning a new concept or simply browsing the Perl Documentation library, you should just use the Web site.

Apache Documentation

The complete Apache documentation is not usually installed alongside the Apache Web server. The best place

to access it is directly from the Apache Web site, <http://httpd.apache.org/docs/2.2/>.

The Apache documentation site groups the content by reference manual, users' guide, tutorials, and platform-specific articles. The format of the documentation implies a steep learning curve, and topics range from moderate to advanced usage.

Unfortunately, the layout and topics covered on this page assume that you are already fairly familiar with the Apache Web server, that you have no problems understanding where to apply changes in the multitude of Apache configuration files, and that you already fully understand all the various related acronyms. The documentation is not well suited for beginning Apache users, and actually discourages a lot of people from using the program.

This book is designed to fill that gap and help the beginning Apache user: however, the online documentation acts as an excellent reference manual. Apache uses hundreds of configuration file directives; if you want to learn more about how a specific one works, where you can use it, and its syntax, consult the "Run-time Configuration Directives" section of the online manual.

Forums

When documentation fails to provide an answer, you can use online forums to ask your peers about the problem. It is paramount to find the right forum to post a question to; posting to the wrong forum does not give you a usable answer, and posting to a sparsely populated forum may result in no one having the expertise to provide an answer.

Once you locate a forum, chances are someone has already encountered your problem in the past and a solution has been proposed. Before you post anything, do a search within the forum's archives for your problem.

Forums can be a frustrating place for new members. Often a developer becomes discouraged because her problem is persistently ignored by the regular user-base. Most of the time, this is caused by not asking the right question, or not providing enough detail about the problem.

A useful read for anyone posting a technical question to a forum is Eric S. Raymond's essay, *How to Ask Questions the Smart Way*, available at

<http://catb.org/~esr/faqs/smart-questions.html>

FORUM NAME	URL
Apache Lounge	www.apachelounge.com
Apache Forums	www.apache.com/forums
PerlMonks	www.perlmonks.org
Perl Guru Forums	www.perlguru.com

Introducing ActivePerl for Windows

Because Perl is an open-source programming language, it is possible for you to select from multiple, customized versions, called *distributions*, of the language for your particular operating system. One popular Perl solution for Windows is called ActivePerl, provided by ActiveState Software Inc.

ActiveState provides a community-supported distribution, called ActivePerl, and a corporate-level distribution, called

ActivePerl License

ActivePerl is licensed by the ActiveState Community License as described at www.activestate.com/activeperl/license. Because ActiveState has packaged the original Perl software, originally produced by Larry Wall, the original Artistic License for Perl has been maintained for all Open Source software content. You can find information on the Artistic License online at www.perl.com/language/misc/Artistic.html. In essence, ActivePerl can be freely used by anyone looking to learn about, utilize, develop, and deploy Perl applications within a corporate or personal environment.

Third-Party Modules

ActivePerl provides its own infrastructure for searching, downloading, and installing third-party modules, called Perl Package Manager (PPM). ActiveState maintains the PPM repositories, keeping them as up-to-date as possible, and providing pre-compiled binary modules sourced from the Comprehensive Perl Archive Network (CPAN) repository.

The ActivePerl PPM program runs as a Windows application, and is compatible with the Perl CPAN program. You can use either program interchangeably to install new Perl modules. You can find information on using PPM and installing third-party modules over PPM in Chapter 9.

Documentation and Support

ActiveState supplies all documentation for all original programs through online resources such as ActiveState Docs, and through the ActivePerl User Guide, which is included with the ActivePerl installation. ActiveState also provides the `perldoc` program, which is a documentation resource built into Perl. Because the PerlDoc utility is available on all Perl distributions, this book will refer you to this utility for more information on a particular function or module.

ActivePerl Enterprise. Although both versions are fully compatible with the core Perl project, the Enterprise distribution provides additional options for professional support and maintenance agreements.

Along with Windows, ActivePerl installation packages are also available for Linux, Mac OS X, and other Unix systems. ActiveState also produces similar packages for the Python and Tcl languages.

ActivePerl for Business

Even though Perl source code is available freely on the Internet, and supported by hundreds of thousands of Perl developers striving to produce the highest-quality product, many corporations require a higher-level service that they can use to support enterprise-level Perl applications and deployments.

ActiveState provides ActivePerl Enterprise for this purpose. Other features include a guaranteed Service Level Agreement (SLA), unlimited support queries, and access to in-house professional developers of ActiveState. You can find more information online at www.activestate.com/business_solutions.

Community Resources

ActiveState hosts a number of services that are free to the ActivePerl community. These include all core Perl documentation, a support forum, mailing lists, FAQs, screencasts, blogs, and third-party code examples. You can find these services on the ActiveState Programmer Network at <http://aspn.activestate.com>.

You can run the `perldoc` program from the DOS prompt. It takes several different arguments, but most of the time, you will only need two. To query a particular function's documentation, run `perldoc -f function`. To query a particular module's documentation, run `perldoc module`. Finally, run `perldoc perldoc` to bring up the help manual. See Appendix A for more information on how to use PerlDoc.

Introducing Strawberry Perl for Windows

Strawberry Perl is a community-supported, free software distribution of Perl that provides all the features of Perl on Unix to Windows systems. Users who are already familiar with Unix Perl but require a Windows development environment should use Strawberry Perl.

The developers who produce and maintain Strawberry Perl recommend their program for intermediate or

advanced Perl users. Beginners are directed to install ActivePerl from ActiveState as no graphical interface is provided by Strawberry Perl.

Strawberry Perl is compatible with Windows XP, Windows Vista, Windows 7, Windows Server 2003 and 2008, and supports both 32-bit and 64-bit OS environments. Strawberry Perl is available online at <http://strawberryperl.com/>.

Strawberry License

Because Strawberry Perl contains various modules and programs, its license is an aggregate of many open-source-compatible licenses, including the GNU General Public License (GPL), published by the Free Software Foundation, and the Perl Artistic License. After installing Strawberry Perl, you can find specific information about the various licenses involved under the directory `C:\strawberry\license\`.

Third-Party Modules

Strawberry Perl is fully compatible with modules delivered through the CPAN repository. Included with the Strawberry Perl installation package is a C compiler and build environment, which can handle any complex CPAN package that generates C code or libraries. You can find information on using CPAN and installing third-party modules over CPAN in Chapter 9.

Strawberry Perl Quirks

Strawberry Perl is a newer distribution of Perl for Windows. Originally based upon the Vanilla Perl Project, the distribution has only been available since January 2008. It strives to adhere as closely as possible to the core Perl fundamentals, but it does have its own way of doing things.

The biggest quirk that you will encounter is its non-standard installation path. This path needs special attention when you start developing CGI code with Strawberry Perl and Apache on Windows. For more information, see Chapter 10.

Documentation and Support

Because Strawberry Perl is a community-supported project, no commercial support options are available. If you require premium-level support options, the Strawberry Perl developers recommend using ActivePerl Enterprise.

Online Resources

Virtually all online resources that are available for Perl also apply to the Strawberry Perl distribution for Windows. The best online source for Perl documentation is <http://perldoc.perl.org>. The primary online resource for the Win32 Perl community is <http://win32.perl.org>. Strawberry Perl does have a support page, which describes its focus and goals, and includes a mailing list and other links, at <http://strawberryperl.com/support.html>. An IRC channel is also available for live support at #win32 on irc.perl.org.

Built-In Perl Documentation

Strawberry Perl does provide a built-in documentation resource for Perl, the `perldoc` program.

Because the `perldoc` utility is available on all Perl distributions, this book will refer you to this utility for more information on a particular function or module.

You can run the `perldoc` program from the DOS prompt. It takes several different arguments, but most of the time, you will only need two. To query a particular function's documentation, run `perldoc -f function`. To query a particular module's documentation, run `perldoc module`.

Finally, run `perldoc perldoc` to bring up the help manual. See Appendix A for more information on how to use PerlDoc.

Download ActivePerl for Windows

ActivePerl is a Perl binary distribution made available by ActiveState. To install it, you need to download a Windows Installer (MSI) package of ActivePerl which provides a Perl interpreter, documentation, and several standard Perl modules that you can use on a Windows system.

ActivePerl is available in two stable versions, ActivePerl 5.8 and ActivePerl 5.10, which are based upon Perl 5.8 and Perl 5.10. Perl 5.10 changed in a way that affected backward-compatibility with some Perl functions, pre-compiled binaries, and existing sample scripts. Some large-scale programs that were written for Perl 5.8 and earlier may require minor code adjustments to run as a Perl 5.10 program. Also, some third-party modules available on CPAN or the ActivePerl PPM Network are not compatible under Perl 5.10.

Download ActivePerl for Windows

- 1 Open a Web browser.
- 2 Type <http://www.activestate.com/activeperl/downloads/> and press Enter.

The ActiveState ActivePerl Web site loads.

- 3 Click the Download ActivePerl link.

ActiveState prompts you to subscribe to the ActiveState Newsletter.

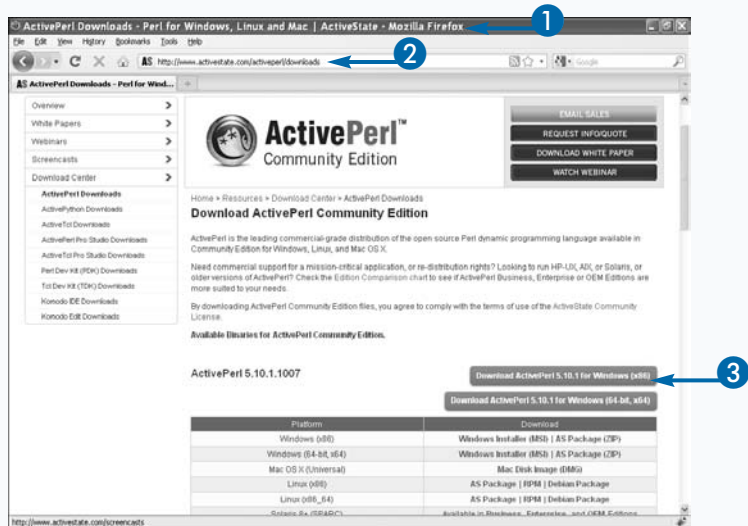
- 4 If you are interested in the ActiveState newsletter, complete the form fields and click Sign Up.

Note: The newsletter sign up is optional, but is available if you are interested in knowing more about how to use ActivePerl.

If you already have a large amount of code written for Perl 5.8, or you require a number of obscure and out-of-date third-party modules, it may be best to stay with ActivePerl 5.8. However, if you are relatively new to Perl and do not have a specific reason to remain on Perl 5.8, you should download Perl 5.10. The ActivePerl download is about 17MB, and uses about 90MB of disk space when installed.

After you finish installing ActivePerl, read the release notes for known incompatibilities between ActivePerl 5.8 and 5.10. Downgrading ActivePerl is as simple as uninstalling version 5.10 and downloading and installing version 5.8.

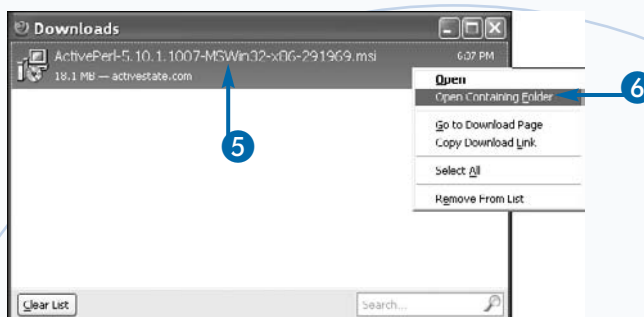
You can download other versions of ActivePerl that are older than version 5.8 from the ActiveState Downloads site at <http://downloads.activestate.com/ActivePerl/>.



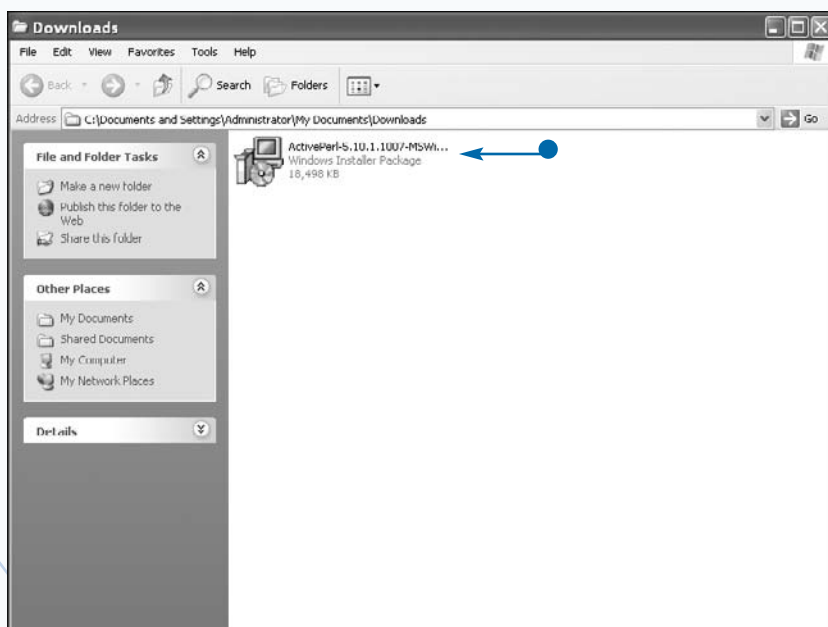
The Downloads dialog box appears, confirming that the download is complete.

- 5 Right-click the ActivePerl download.

- 6 Click Open Containing Folder.



- The ActivePerl installation package appears in the folder you saved the download into.



Extra

Additional tools for ActivePerl are available for download on the ActiveState site. The Komodo Integrated Development Environment, or IDE, provides a multi-language editor, debugger, and front-end to various source-code managers such as Concurrent Versions System (CVS) and Subversion. The Perl Dev Kit, or PDK, provides a set of critically acclaimed tools for developing large-scale Perl applications.

ActivePerl Pro Studio provides both the IDE and PDK, as well as access to Safari Books Online, an electronic library of thousands of technical publications.

The Komodo IDE and the PDK are commercial applications that require you to purchase a license from ActiveState. ActivePerl Pro Studio requires a yearly subscription.

This process of downloading ActivePerl for Windows assumes that the browser is on the same computer where you will install ActivePerl. If this is not the case, you can manually select an ActivePerl download for alternative Operating System environments.

It is also possible to download older and experimental releases of ActivePerl on this page.

Install ActivePerl for Windows

You must install the ActiveState ActivePerl installation package onto each Windows workstation or server that you will be developing Perl CGI scripts on, or servicing Perl CGI queries online. The supported versions of Windows for ActivePerl 5.10.1 are Windows XP, Windows Vista, and Windows Server 2003 and 2008.

The installation of ActivePerl for Windows is handled by a Windows Installer Package with an .msi extension. The installation process requires administrative privileges on the system. As a result, the user performing the installation must be a member of the local Administrators security group.

In general, the ActivePerl default options are the best choices for new users. The recommended default folder where ActivePerl will be installed is `C:\Perl`.

The installation process prompts you for a series of custom setup options. One set of options confirms whether you want to add Perl to the PATH environment variable, and whether you want to create Perl file extension association. It is important to leave both of these options enabled, which is the default setting.

The size of a new ActivePerl installation is about 90MB, but as other programs are used, such as PPM and CPAN, the overall footprint can grow to 250MB or more.

Install ActivePerl for Windows

- 1 Open the folder containing the ActivePerl download.
- 2 Double-click the ActivePerl installer.
- 3 If a security-warning message appears, click Run.

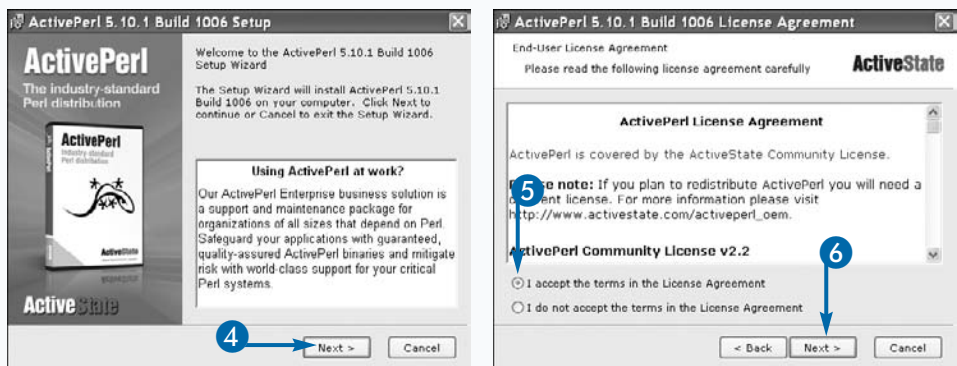


The ActivePerl Setup window opens.

- 4 Click Next.

The ActivePerl License Agreement window opens.

- 5 After reading the license, click the option to accept the terms of the license agreement.
- 6 Click Next.



The Custom Setup screen appears.

- 7 Click Next to continue.

The Choose Setup Options window appears.

Note: Do not uncheck the first two check boxes. These selections are required for proper CGI Perl script execution.

- 8 Click Next.

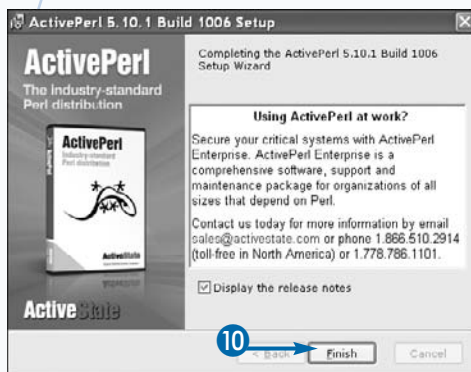
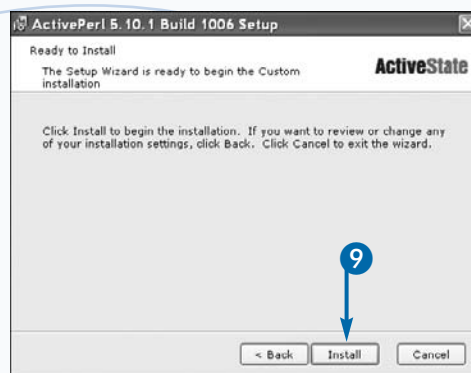
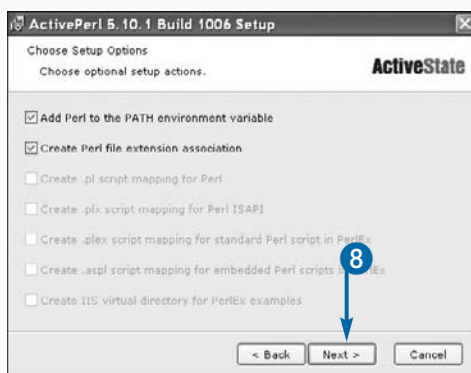
The Ready to Install screen appears.

- 9 Click Install.

A window appears, stating the Setup Wizard is complete.

- 10 Click Finish.

The ActivePerl Release Notes appear.



Extra

Once you complete the installation, ActivePerl becomes available globally in the system path. To confirm that everything is installed correctly, open up a Command Prompt by clicking Start → Programs → Accessories → Command Prompt, and run the Perl interpreter's `version` command.

TYPE THIS

```
perl -v
```



RESULTS

```
This is perl, v5.10.1 built for MSWin32-x86-multi-thread
(with 2 registered patches, see perl -V for more detail)

Copyright 1987-2009, Larry Wall
```

It should not matter what directory you type the `perl -v` command in; after you complete the installation, the Perl interpreter should be available globally in the system path. If Windows reports that this command is invalid, log out of the Windows session and log back in again.

By default, the installation's home directory path is `C:\Perl`. You can change and customize this path during the installation procedure. If you do customize this path, be aware that your CGI scripts will need to reflect the new path. For information on how to do this, see Chapter 10.

Download Strawberry Perl for Windows

Strawberry Perl is a Perl binary distribution made available by the open-source community. To install it, you need to download either a Windows Installer (MSI) package or a Zip file of Strawberry Perl, which provides a Perl interpreter, documentation, and several standard Perl modules that can be used on a Windows system.

Strawberry Perl is available in two stable versions, Strawberry Perl 5.8 and Strawberry Perl 5.10. Perl 5.10 changed in a way that affected backward-compatibility with some Perl functions, pre-compiled binaries, and existing sample scripts. Some large-scale programs that were created with Perl 5.8 and earlier may require some code adjustments to run as a Perl 5.10 program. Also, some third-party modules available on CPAN have not yet been updated to be compatible under Perl 5.10.

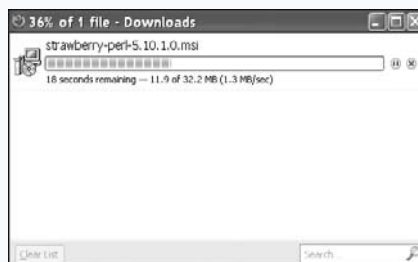
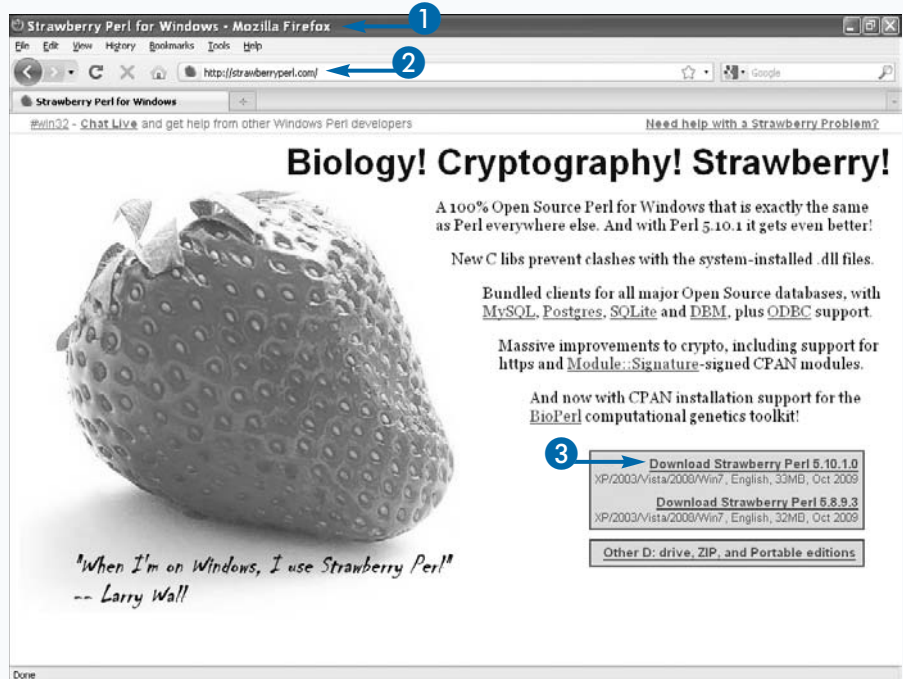
Download Strawberry Perl for Windows

- 1 Open a Web browser.
- 2 Type **http://strawberryperl.com** and press Enter.
The Strawberry Perl Web site loads.
- 3 Click the Download Strawberry Perl link.

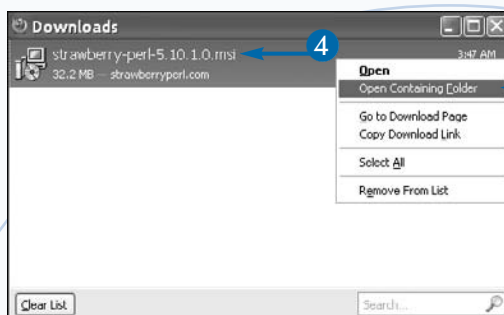
If you use Firefox, the Downloads dialog box appears, confirming that the download is complete.

The Strawberry Perl download is about 32MB, and uses about 135MB of disk space when installed. The actual installation procedure is very simple. There are no options to customize the installation directory or components. Instead, Strawberry Perl produces three separate download packages which customize where Strawberry Perl is installed: a standard MSI package, a ddrive MSI package, and a Zip file.

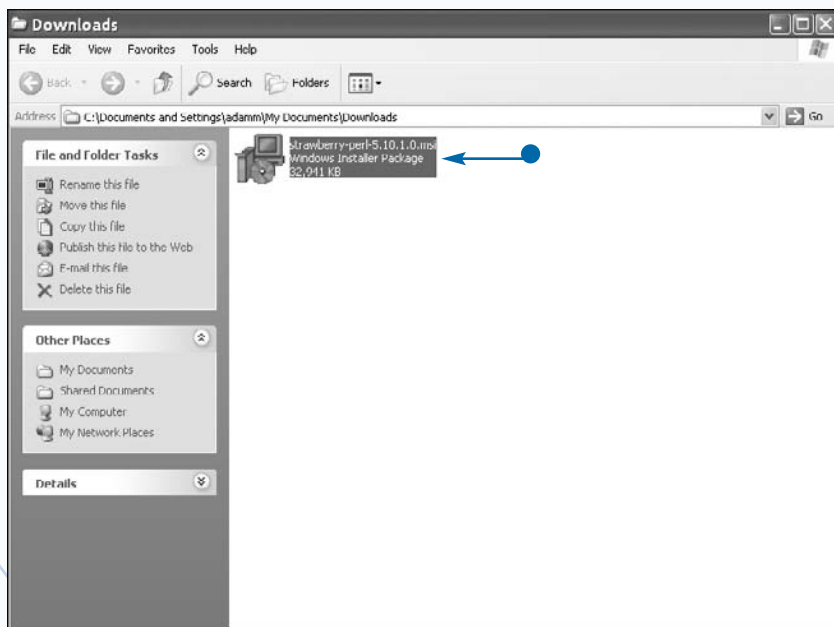
Both the standard and ddrive MSI packages will install Strawberry Perl onto your Windows workstation or server; however, the standard version will install Perl into the directory `C:\strawberry`, and the ddrive into `D:\strawberry`. The Zip file allows you to extract the Strawberry Perl binaries into a directory path of your choosing. You can download these alternative versions of Strawberry Perl from the Strawberry Perl Releases page at <http://strawberryperl.com/releases.html>.



- 4 Right-click the Strawberry Perl download.
- 5 Click Open Containing Folder.



- The downloaded Strawberry Perl installation package appears in the folder you saved the download into.



Extra

The Strawberry Perl download delivers an MSI package onto your workstation. You need to install this MSI onto every Windows machine that will be servicing Perl CGI requests with Strawberry Perl.

Two other download packages are available from Strawberry Perl as alternatives to the standard MSI: a ddrive MSI and a Zip file. Which one you use depends on the directory where you want Strawberry Perl to be installed.

The standard MSI installs into `C:\strawberry\perl` by default, the ddrive MSI uses `D:\strawberry\perl`, and the Zip file allows you to manually extract the distribution binaries into a directory of your choosing.

If you do choose the Zip file download and installation method, you need to manually set up the system path to reflect the custom install directory.

Install Strawberry Perl for Windows

You must install the Strawberry Perl binary packages onto each Windows workstation or server that you will be developing Perl CGI scripts on, or servicing Perl CGI queries online. The supported versions of Windows for Strawberry Perl 5.10.1.0 are Windows XP, Windows Vista, Windows 7, Windows Server 2003 and Windows Server 2008.

The installation of Strawberry Perl for Windows is normally handled by a Windows Installer Package with an .msi extension, but alternative installation methods are available. The MSI installation process requires administrative privileges on the system. As a result, the user performing the installation must be a member of the local Administrators security group.

The Strawberry Perl installation program does not provide any options to customize the installation process. If you

downloaded the standard MSI package, the default folder where Strawberry Perl is installed is `C:\strawberry`. If you downloaded the ddrive MSI package, the default folder will be `D:\strawberry`. If you want a custom directory path, you may extract the Zip file archive into any directory you choose.

If you use the MSI method, Strawberry Perl is available automatically in the global system path. If you install Strawberry Perl by extracting the Zip file, you need to manually set up the system path to your custom installation directory. You must make sure that the following directories are added to your path:

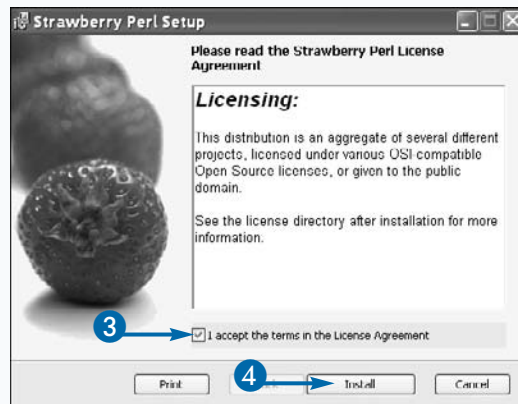
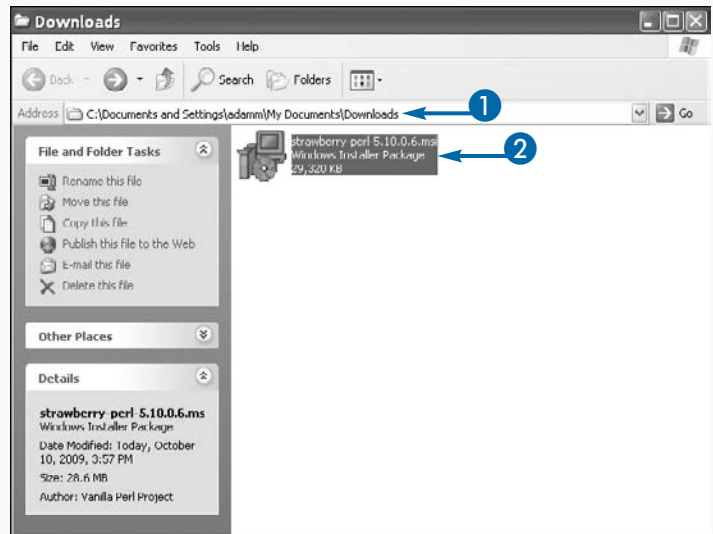
`C:\extractpath\c\bin`

`C:\extractpath\perl\bin`

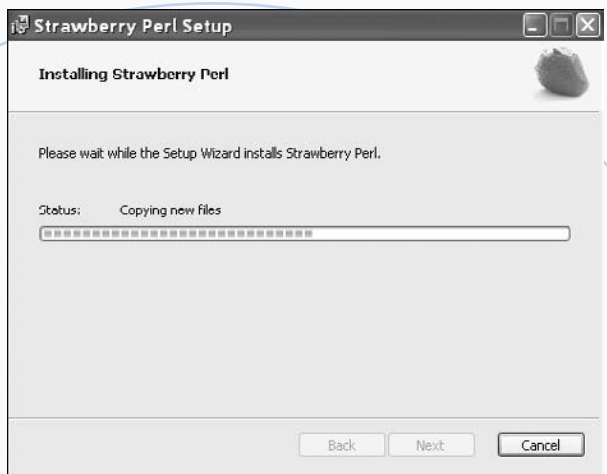
The size of a new Strawberry Perl installation is about 130MB, but as other programs are used, such as CPAN, the overall footprint can grow to 250MB or more.

Install Strawberry Perl for Windows

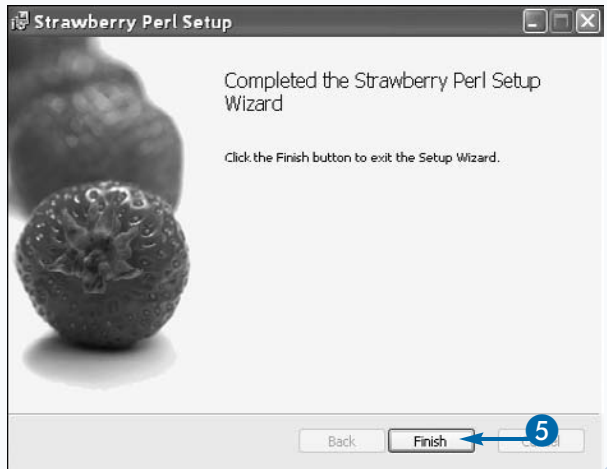
- 1 Open the folder containing the downloaded Strawberry Perl installation package.
- 2 Double-click the Strawberry Perl installer.
- 3 After reading the license, click the option to accept the License Agreement.
- 4 Click Install.



The Installing Strawberry Perl window appears, showing the installation process.



A window appears, stating that the Setup Wizard is complete.



- 5 Click Finish.

Extra

To confirm that everything is installed correctly, open up a Command Prompt and attempt to run the Perl interpreter's version command.

TYPE THIS

```
perl -v
```



RESULTS

```
This is perl, v5.10.1 built for MSWin32-x86-multi-thread
Copyright 1987-2007, Larry Wall
```

It should not matter what directory you type the `perl -v` command in; it must be available globally in the system path. If Windows reports that this command is invalid, log out of the Windows session and log back in again.

The installation's default home directory path is `C:\strawberry\perl`. You can change this by installing the ddrive MSI package, or extracting the Zip file. If you customize this path, your CGI scripts will need to reflect the new path (see Chapter 10).

If you manually install the Zip file, remember to add Strawberry Perl into the global system path. Right-click My Computer, click Advanced, click Environment Variables, and then edit the `Path` system variable.

Install Perl for Debian/Ubuntu Linux

The Perl interpreter and core modules are installed by default on all Debian- and Ubuntu-based Linux systems; however, the Perl documentation package is usually not. You can download the Perl documentation onto your Debian or Ubuntu Linux system. If you do not require the Perl documentation program, you can skip this section.

Debian and Ubuntu systems use the DEB packaging format. You can use the program suite called Advanced Package Tool, or APT, for downloading, installing, upgrading, and removing DEB packages. APT is standard on all Debian and Ubuntu systems, for more information see www.debian.org/doc/user-manuals.

The APT repository that houses the Perl DEB packages tends to only contain the latest stable Perl release milestone. If you need an earlier release, such as Perl 5.8, you can download and install it from the Perl project homepage. You can also use the ActiveState 5.8 Linux DEB package.

The installation process requires that your computer be connected to the Internet. The download, installation, and setup procedure is handled by a single command, `apt-get`, which you execute in a Terminal window.

The Perl download is about 8.6MB, and uses about 32MB of disk space when installed. To open a Terminal window in a Debian or Ubuntu system and launch an APT command, click the Applications button, go to the Accessories menu, and then click Terminal.

Install Perl for Debian/Ubuntu Linux

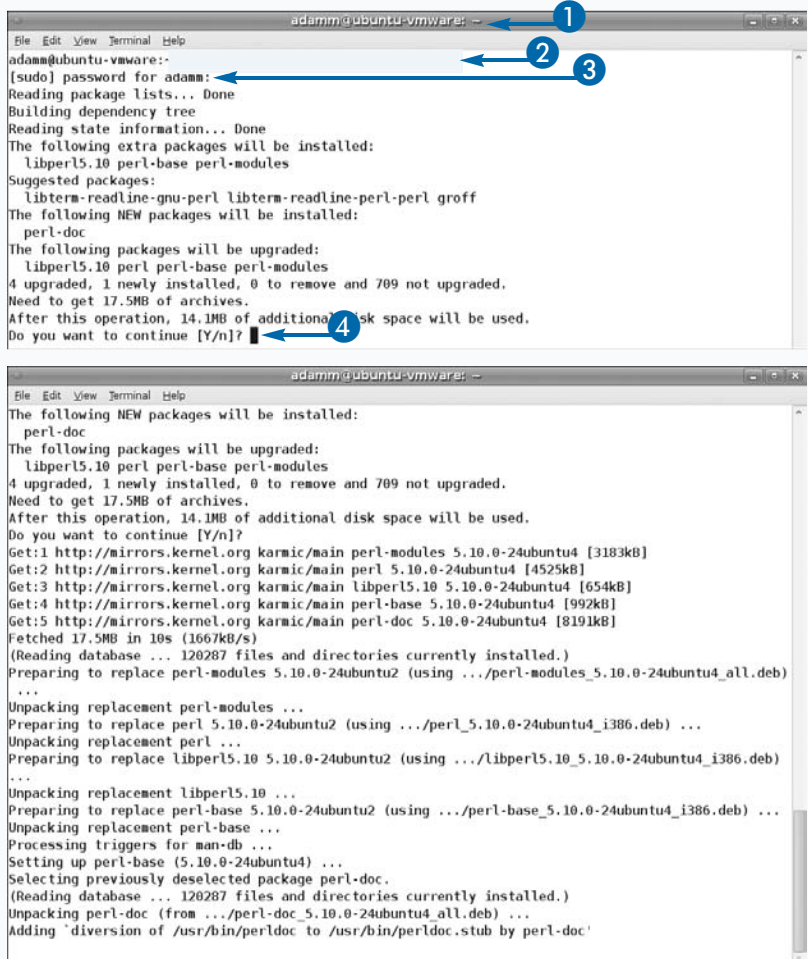
- 1 Open a Terminal window on Debian or Ubuntu.
- 2 Type `sudo apt-get install perl perl-doc` and press Enter.

Note: In Debian, the `sudo` command may not work. Instead type `su -c 'apt-get install perl perl-doc'` to run the command as root.

- 3 Type your password if prompted.
- 4 Type `Y` and press Enter to continue.

The Perl interpreter and Perl documentation are downloaded and installed.

Note: This process also upgrades Perl, if an upgrade is available on the APT repository.



```
adam@ubuntu-vmware: ~$ sudo apt-get install perl perl-doc
[sudo] password for adam:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libperl5.10 perl-base perl-modules
Suggested packages:
  libterm-readline-gnu-perl libterm-readline-perl groff
The following NEW packages will be installed:
  perl-doc
The following packages will be upgraded:
  libperl5.10 perl perl-base perl-modules
4 upgraded, 1 newly installed, 0 to remove and 709 not upgraded.
Need to get 17.5MB of archives.
After this operation, 14.1MB of additional disk space will be used.
Do you want to continue [Y/n]? Y

Get:1 http://mirrors.kernel.org/karmic/main perl-modules 5.10.0-24ubuntu4 [3183kB]
Get:2 http://mirrors.kernel.org/karmic/main perl 5.10.0-24ubuntu4 [4525kB]
Get:3 http://mirrors.kernel.org/karmic/main libperl5.10 5.10.0-24ubuntu4 [654kB]
Get:4 http://mirrors.kernel.org/karmic/main perl-base 5.10.0-24ubuntu4 [992kB]
Get:5 http://mirrors.kernel.org/karmic/main perl-doc 5.10.0-24ubuntu4 [8191kB]
Fetched 17.5MB in 10s (1667kB/s)
(Reading database ... 120287 files and directories currently installed.)
Preparing to replace perl-modules 5.10.0-24ubuntu2 (using .../perl-modules_5.10.0-24ubuntu4_all.deb)
...
Unpacking replacement perl-modules ...
Preparing to replace perl 5.10.0-24ubuntu2 (using .../perl_5.10.0-24ubuntu4_i386.deb) ...
Unpacking replacement perl ...
Preparing to replace libperl5.10 5.10.0-24ubuntu2 (using .../libperl5.10_5.10.0-24ubuntu4_i386.deb)
...
Unpacking replacement libperl5.10 ...
Preparing to replace perl-base 5.10.0-24ubuntu2 (using .../perl-base_5.10.0-24ubuntu4_i386.deb) ...
Unpacking replacement perl-base ...
Processing triggers for man-db ...
Setting up perl-base (5.10.0-24ubuntu4) ...
Selecting previously deselected package perl-doc.
(Reading database ... 120287 files and directories currently installed.)
Unpacking perl-doc (from .../perl-doc_5.10.0-24ubuntu4_all.deb) ...
Adding diversion of /usr/bin/perldoc to /usr/bin/perldoc.stub by perl-doc'
```

Install Perl for Red Hat Linux

The base Perl package is installed by default on all Red Hat-based Linux systems. This includes the Perl interpreter, documentation, and core modules. If you run these commands on a system that already has Perl installed, and the system detects that a newer version is available online, Perl is automatically upgraded. Linux machines that run on Red Hat-based Linux, including Fedora Linux, CentOS Linux, and Red Hat Enterprise Linux, all use the Red Hat Package Manager (RPM) packaging format. Red Hat has standardized on the Yellowdog Updater, Modified (YUM) program for downloading, installing, and removing RPMs.

The Red Hat repository that houses the Perl RPM packages tends to only contain the latest Perl release milestone. If you need an earlier release, such as

Perl 5.8, you can download and install it from the Perl project homepage.

The installation process requires that your computer be connected to the Internet. The download, installation, and setup procedure is handled by a single command, `yum`, which you execute in a Terminal window.

On the command-line, alternatives to `yum` on Red Hat-based Linux systems include running the `urpmi` command and downloading the RPM file. If you are using a KDE graphical interface, you can use the KPackage program, or if you use a Gnome graphical interface, the GnoRPM and Gnome Package Kit programs are available. The Perl download is about 8.7MB, and uses about 30.4MB of disk space when installed. To open a Terminal window in a Red Hat system, click the Main Menu button, go to the System Tools menu (System in KDE; Accessories in Fedora), and click Terminal.

Install Perl for Red Hat Linux

- 1 Open a Terminal window on Red Hat.
- 2 Type `sudo yum install perl` and press Enter.
- 3 Type your password and press Enter.
- 4 Type `Y` and press Enter.

Note: If `sudo` is not available on your system, type `su -c 'yum install perl'` instead. Then type `root's` password.

The Perl RPM package is downloaded and installed.

Note: This process also upgrades Perl, if an upgrade is available on the RPM repository.

```

adammm@redhat ~$ sudo yum install perl
[sudo] password:
Package      Arch      Version      Repository      Size
-----
perl         i386      4:5.8.8-18.el5_3.1 updates      12 M

Transaction Summary
  Install      0 Package(s)
  Update       1 Package(s)
  Remove       0 Package(s)

Total download size: 12 M
Is this ok [y/N]:
  
```

```

adammm@redhat ~$ sudo yum install perl
[sudo] password:
Package      Arch      Version      Repository      Size
-----
perl         i386      4:5.8.8-18.el5_3.1 updates      12 M

Transaction Summary
  Install      0 Package(s)
  Update       1 Package(s)
  Remove       0 Package(s)

Total download size: 12 M
Is this ok [y/N]: y
Downloading Packages:
(1/1): perl-5.8.8-18.el5_3.1.rpm 100% |=====| 12 MB  00:01
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating : perl                               [1/2]
  Cleanup  : perl                               [2/2]

Updated: perl-5.8.8-18.el5_3.1
Complete!
adammm@redhat ~$
  
```


Download ActivePerl for Linux or Unix

ActivePerl is a Perl binary distribution that is made available by ActiveState. To install it, you must download an ActivePerl installation package, which is available as a gzipped tarball, a Debian Package, or in a Red Hat Package Manager file format. Each package provides a Perl interpreter, documentation, and several standard Perl modules that you can use on a Linux system. This section is specific to the download of the gzipped tarball, or `tar.gz`, installation package. If you are running on a Debian/Ubuntu or Red Hat system, you should download the appropriate Debian or Red Hat package, respectively.

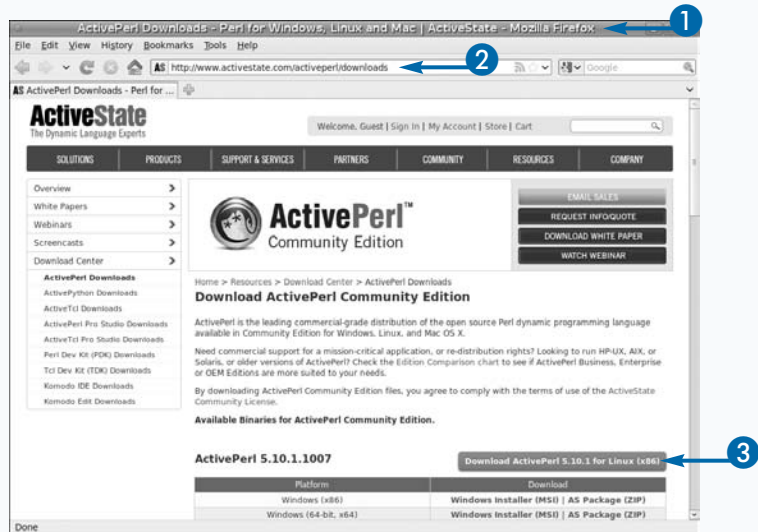
ActivePerl is available in two stable versions, ActivePerl 5.8 and ActivePerl 5.10, which are based upon Perl 5.8 and Perl 5.10, respectively. Perl 5.10 changed in a way that affected

backward-compatibility with some Perl functions, pre-compiled binaries, and existing sample scripts. Some large-scale programs that were written for Perl 5.8 and earlier may require minor code adjustments to run as a Perl 5.10 program. Also, some third-party modules available on the Comprehensive Perl Archive Network (CPAN) or the ActivePerl Perl Package Manager (PPM) Network have not yet been updated to be compatible under Perl 5.10.

The ActivePerl download is about 30MB, and uses about 115MB of disk space when installed. After the installation of ActivePerl is complete, read the release notes for known incompatibilities between ActivePerl 5.8 and 5.10. Downgrading ActivePerl is as simple as uninstalling version 5.10 and downloading and installing version 5.8. You can download other versions of ActivePerl that are older than version 5.8 from the ActiveState Downloads site at <http://downloads.activestate.com/ActivePerl/>.

Download ActivePerl for Linux or Unix

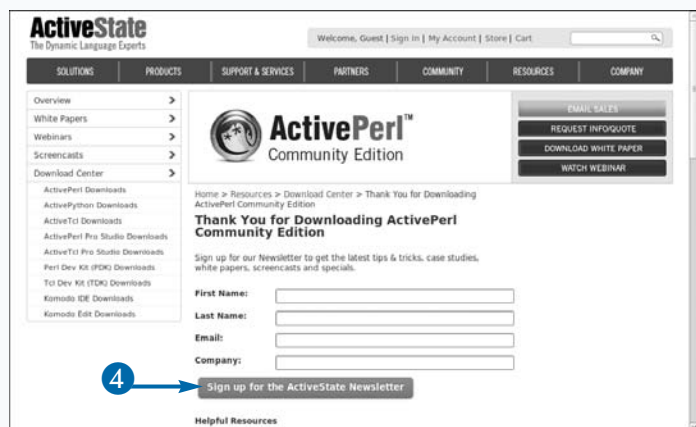
- 1 Open a Web browser.
 - 2 Type **`www.activestate.com/activeperl/downloads/`** and press Enter.
- The ActiveState ActivePerl Web site loads.
- 3 Click the Download ActivePerl link.



ActiveState prompts you to subscribe to the ActiveState Newsletter.

- 4 If you are interested in the newsletter, complete the form fields and click Sign up for the ActiveState Newsletter.

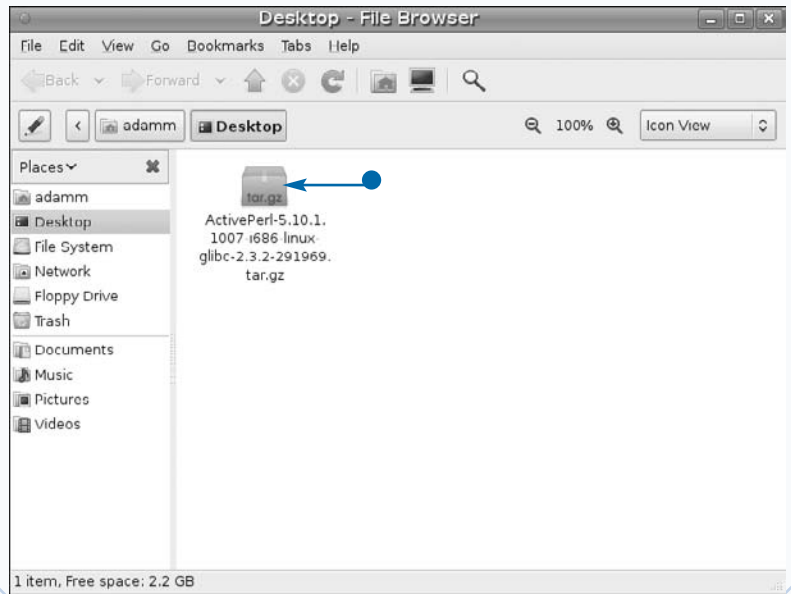
Note: The newsletter sign up is optional, but is available if you are interested in knowing more about how to use ActivePerl.



- 5 After the download is complete, right-click the ActivePerl file.
- 6 Click Open Containing Folder.



- The ActivePerl Installation Package appears in the folder you saved the download into.



Apply It

The process of downloading ActivePerl for Linux delivers a `tar.gz` file to your computer. If you are running a Debian- or Red Hat-based system, you should download the ActivePerl DEB or RPM package format instead.

To download a DEB or RPM package, or to select an alternative Unix platform, select the binary package on the ActiveState ActivePerl downloads Web page. It is also possible to download an older release of ActivePerl on this page.

PLATFORM	DOWNLOAD
Linux (x86)	TGZ Archive, RPM, or Debian Package
Linux (64-bit, x64)	TGZ Archive, RPM, or Debian Package
Solaris 8+ (SPARC)	TGZ Archive, Solaris Package
Solaris 8+ (SPARC 64-bit)	TGZ Archive, Solaris Package
Solaris 8+ (x86)	TGZ Archive, Solaris Package
AIX 5 (PowerPC)	TGZ Archive

Install ActivePerl for Linux or Unix

You must install the ActiveState ActivePerl binary package onto each Linux workstation or server, that you will be developing Perl CGI scripts on, or servicing Perl CGI queries online. Installing ActivePerl onto your Linux workstation rather than the standard Perl distribution gives you access to the ActiveState PPM network and its commercial development programs.

An installer shell script handles the installation of ActivePerl for Linux when you use a gzipped tarball, or `tar.gz`, package archive. The installation process requires root privileges on the system. Usually `sudo` or `su` can be used by a regular user account to gain root privileges.

Once installed, the ActiveState Perl binary becomes available in tandem with your system's native Perl binary, if one already exists. The default installation directory is `/opt/ActivePerl-5.10/`, assuming you downloaded

the 5.10 version. It is a very good idea to set up a symbolic link from the default Perl interpreter location, `/usr/bin/perl`, to the ActiveState Perl interpreter binary. This process is explained at the end of this section.

This section is only intended for the `tar.gz` download-and-installation method of ActivePerl for Linux. You should only use this method on Linux or Unix systems that are not based on Red Hat or Debian. With these systems, you only need to double-click the ActivePerl DEB or RPM package to install the program.

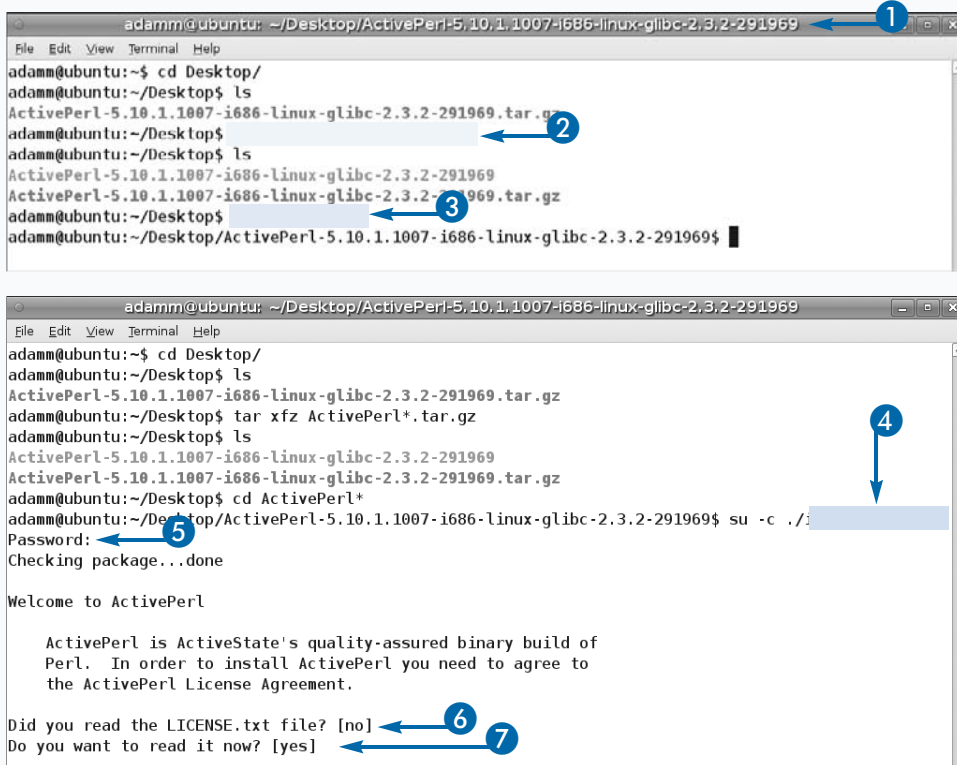
In place of using the command-line, some graphical window managers may allow you to double-click the `tar.gz` file. That is sufficient to uncompress the downloaded archive, and skip steps 1 to 3, but you still need to execute the `install.sh` script with root privileges.

Install ActivePerl for Linux or Unix

- 1 Open a Terminal window in the directory where you downloaded the ActivePerl `tar.gz` file.
- 2 Type `tar xzf ActivePerl*.tar.gz` to uncompress the archive.
- 3 Type `cd ActivePerl*` and press Enter.
- 4 Type `su -c ./install.sh` and press Enter to begin the installation as root.
- 5 Type the root account's password and press Enter.

Note: If you do not know root's password, replace `su -c` with `sudo`. If that fails, you can run the installation as a normal user, but you need to select a different path in step 9.

- 6 Press Enter to confirm that you have not yet read the license file.
- 7 Press Enter to read the license file.



```
adammm@ubuntu: ~/Desktop/ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969
File Edit View Terminal Help
adammm@ubuntu:~$ cd Desktop/
adammm@ubuntu:~/Desktop$ ls
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969.tar.gz
adammm@ubuntu:~/Desktop$ tar xzf ActivePerl*.tar.gz
adammm@ubuntu:~/Desktop$ ls
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969.tar.gz
adammm@ubuntu:~/Desktop$ cd ActivePerl*
adammm@ubuntu:~/Desktop/ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969$

adammm@ubuntu: ~/Desktop/ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969
File Edit View Terminal Help
adammm@ubuntu:~$ cd Desktop/
adammm@ubuntu:~/Desktop$ ls
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969.tar.gz
adammm@ubuntu:~/Desktop$ tar xzf ActivePerl*.tar.gz
adammm@ubuntu:~/Desktop$ ls
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969
ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969.tar.gz
adammm@ubuntu:~/Desktop$ cd ActivePerl*
adammm@ubuntu:~/Desktop/ActivePerl-5.10.1.1007-i686-linux-glibc-2.3.2-291969$ su -c ./install.sh
Password:
Checking package...done

Welcome to ActivePerl

ActivePerl is ActiveState's quality-assured binary build of
Perl. In order to install ActivePerl you need to agree to
the ActivePerl License Agreement.

Did you read the LICENSE.txt file? [no]
Do you want to read it now? [yes]
```


The ActivePerl License Agreement appears. You may press the spacebar to page down the license text, or type **q** to quit reading.

- 8 Type **yes** to agree to the license and then press Enter.
- 9 Press Enter to select the default top-level directory.
- 10 (Optional) Press Enter to install HTML documentation.
- 11 Press Enter to proceed with the installation.

- ActivePerl is installing.
- ActivePerl is now installed.

```

adam@ubuntu: ~/Desktop/ActivePerl-5.10.1.1007-1686-linux-glibc-2.3.2-291969
file Edit View Terminal Help

foreign agency or authority, and will not distribute, export or
re-export, or allow the distribution, export or re-export, of the
Package or any copy or adaptation of direct product thereof, or any
underlying technology, except in full compliance with any and all
such applicable laws, restrictions and regulations. You represent
and warrant that you are not located in, under the control of, or a
national or resident of, any restricted country (currently
including Myanmar (Burma), Belarus, Cuba, Libya, North Korea, Iran,
Iraq, Sudan, Syria, and Afghanistan) or of any designated entity or
person.

Do you agree to the ActivePerl License Agreement? [no] yes
8

This installer can install ActivePerl in any location of your
choice. You do not need root privileges. However, please make sure
that you have write access to this location.

Enter top level directory for install? [/opt/ActivePerl-5.10]
9

The ActivePerl documentation is available in HTML format. If installed
it will be available from file:///opt/ActivePerl-5.10/html/index.html.
If not installed you will still be able to read all the basic perl and
module documentation using the man or perldoc utilities.

Install HTML documentation [yes]
10
Ok.

The typical ActivePerl software installation requires
120 megabytes. Please make sure enough free space is available
before continuing.

Proceed? [yes]
11
Ok.

adam@ubuntu: ~/Desktop/ActivePerl-5.10.1.1007-1686-linux-glibc-2.3.2-291969
file Edit View Terminal Help

If not installed you will still be able to read all the basic perl and
module documentation using the man or perldoc utilities.

Install HTML documentation [yes]
Ok.

The typical ActivePerl software installation requires
120 megabytes. Please make sure enough free space is available
before continuing.

Proceed? [yes]
Ok.

Installing ActivePerl...
Copying files to /opt/ActivePerl-5.10...done
Relocating...done (186 files relocated)
Generating HTML documentation...done
Syncing perl PPM database with .packlists...done

ActivePerl has been successfully installed at /opt/ActivePerl-5.10.

Please modify your startup environment by adding:

/opt/ActivePerl-5.10/site/bin:/opt/ActivePerl-5.10/bin to PATH
/opt/ActivePerl-5.10/site/man:/opt/ActivePerl-5.10/man to MANPATH

For general questions or comments about ActivePerl, please
contact us at <support@activestate.com>.

Thank you for using ActivePerl!

adam@ubuntu: ~/Desktop/ActivePerl-5.10.1.1007-1686-linux-glibc-2.3.2-291969

```

Extra

Once the installation is complete, the installer asks you to modify your startup environment. You can usually accomplish this globally by appending the following text to the file `/etc/profile`.

```
PATH="/opt/ActivePerl-5.10/site/bin:/opt/ActivePerl-5.10/bin:$PATH"
MANPATH="/opt/ActivePerl-5.10/site/man:/opt/ActivePerl-5.10/man:$MANPATH"
```

Be sure to check your system's shell manual for specific instructions for manipulating `PATH` and `MANPATH`.

It is generally a good idea to set up `/usr/bin/perl` as a symbolic link to the ActiveState Perl interpreter. Be sure to back up your original Perl binary if one already exists in this path.

```
mv /usr/bin/perl /usr/bin/perl.bak
ln -s /opt/ActivePerl-5.10/bin/perl /usr/bin/perl
```

ActiveState Perl is also available as a DEB and RPM package. This method is preferable over using the `tar.gz` format on Debian- and Red Hat-based systems, as installation, upgrading, and removal is handled by existing infrastructure. In most cases, you can install a DEB or RPM package simply by double-clicking the package file. Alternatively, you can install the package using a Terminal window. On Debian, use the command `sudo dpkg -i packagefile.deb`; on Red Hat systems, use `sudo rpm -i package.rpm` or `su -c 'rpm -i package.rpm'`.

Download Apache for Windows

The Apache Web server is the program that listens and responds to incoming HTTP requests. It is a vital part of presenting dynamic CGI scripts on the Internet. You can download the latest version of the Apache program from the Apache Software Foundation's Web site at <http://httpd.apache.org>. When you download Apache for Windows, the software is packaged as a Window Installer (MSI) file.

Apache runs on all recent Windows operating systems, including workstations running Windows 95, 98, 2000, and XP. However, you should only install Apache onto one of these versions for testing and development purposes. When you are ready to deploy your work onto the Internet, be sure to use Apache on a Windows Server 2003 or 2008 computer, or even a professionally managed Linux server.

Download Apache for Windows

- 1 Open a Web browser.
- 2 Type **http://httpd.apache.org** and press Enter.
- 3 Click the Download from a mirror link.

- 4 Scroll down to the best available version.
- 5 Click the Win32 Binary download.

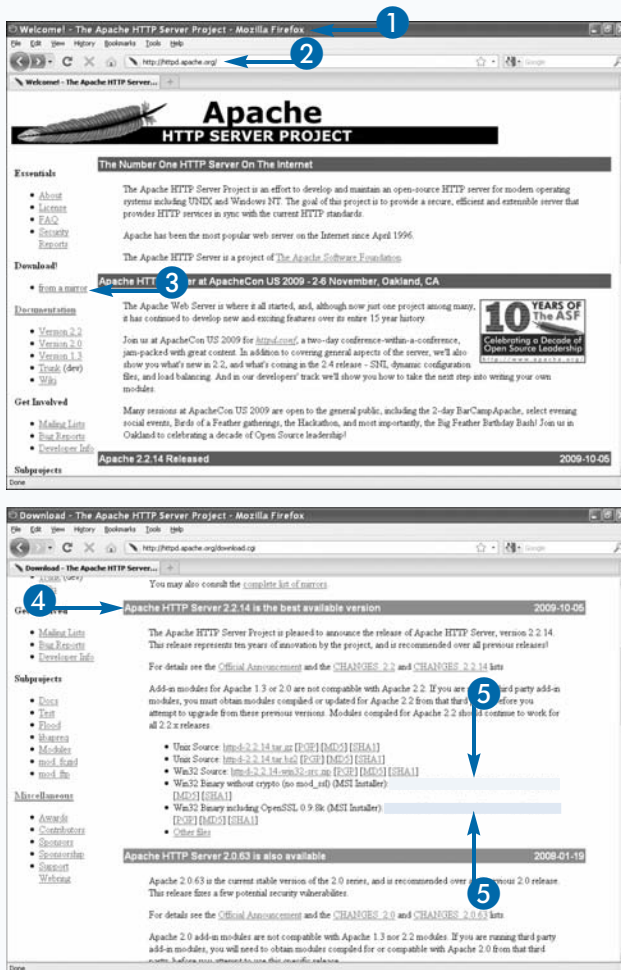
Note: Select the appropriate link to download Apache with or without OpenSSL included.

- 6 Click the Save File button when prompted by your Web browser's pop-up.

The file downloads to your computer.

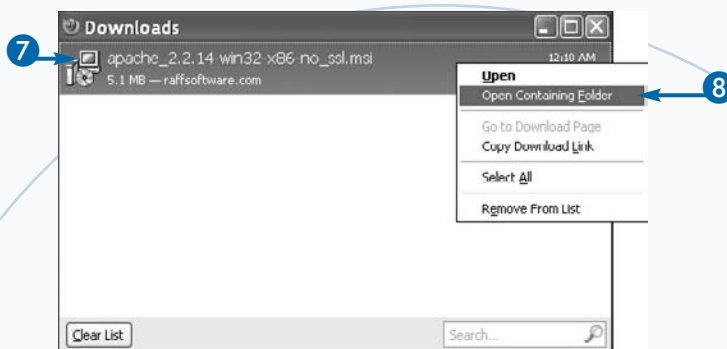
Before you begin downloading Apache, you need to decide which of the two binary packages you want to use. One has the OpenSSL Cryptographic Library included, and the other does not. You are presented with a choice in the form of an offer to comply with strict laws regarding importing, exporting, and possessing certain encryption methods, depending on your jurisdiction. The Apache Software Foundation strongly recommends that all installers of Apache validate if encryption software is permitted in their country; for this you can use the Web site www.wassenaar.org.

If you know that you will not need Secure Socket Layer (SSL) encryption support, you can download the binary without OpenSSL support. If you later decide that you need SSL, you can always download the binary that includes OpenSSL and re-install Apache.

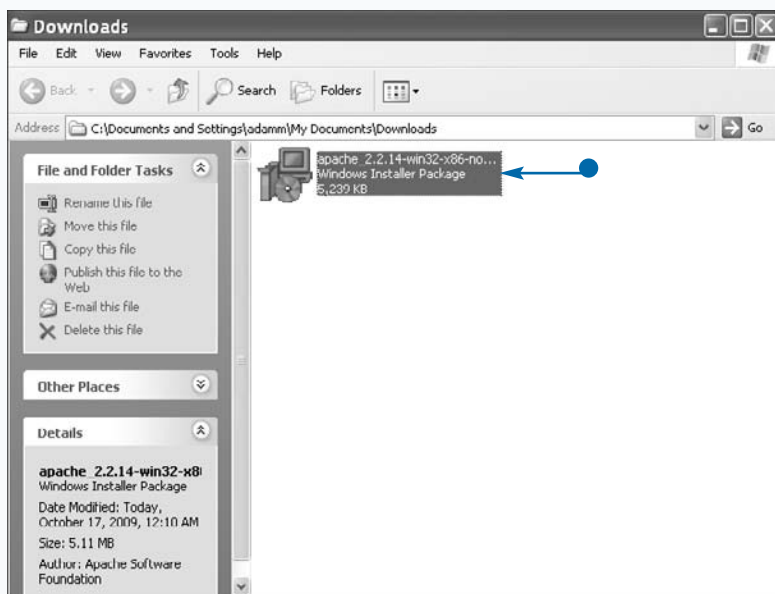


7 Right-click the Apache download.

8 Click Open Containing Folder.



- The downloaded Apache Installation Package appears in your Downloads folder.



Extra

Apache provides a separate Windows download that supports SSL encryption through the OpenSSL library. You can use this alternative download version if you require Apache Web server to handle SSL encryption on your Web site.

The version of Apache for Windows that has OpenSSL encryption built in is subject to strict import and export laws, depending on the country you live in. The U.S. Department of Commerce has classified Apache with OpenSSL encryption as an Export Commodity Control, which restricts which countries can receive this type of software. Also, some countries restrict the import of such software. The Apache Software Foundation strongly advises that you check your country's laws, regulations, and policies before using any type of encryption software.

If you are in doubt, download the version of Apache without built-in SSL encryption support.

If you do require SSL support, you also need to set up a private key and a signed certificate files, and configure your Apache Web server to use them. This process is described in Chapter 22.

Install Apache for Windows

You need to install Apache onto at least one computer before you can start any Perl CGI development. Even if your workstation is not directly connected to the Internet, it is still an essential component for creating any CGI content on your computer.

You can always make HTTP requests to your local Web server using the special hostname, *localhost*. This is actually an alias to the IP address 127.0.0.1, which always acts as the address to the computer you are actively working on. Once you have installed Apache, you can access all programs and files in a browser using the local Web site `http://localhost`.

In a production deployment, you should install Apache onto every server that will be receiving an HTTP request from the Internet. Generally, it is best to hire a professional management service that hosts and maintains a shared bank of Apache servers, for your Web

site. Larger-scale deployments may require a dedicated server, or a group of dedicated servers, each with Apache installed. For more information on finding a hosting provider that is compatible with Perl and Apache, see Chapter 1.

The Windows installation wizard displays several screens, which guide you through the installation process. These include a screen to accept the Apache license, and another screen that requests the server information for this Apache server instance. You need to populate your network domain name, your server's fully qualified name, an Administrator e-mail address, and select a HTTP port number.

In a production environment, Apache uses this information to identify itself when connected to the Internet. If this is a private test server, the default values on this screen are fine.

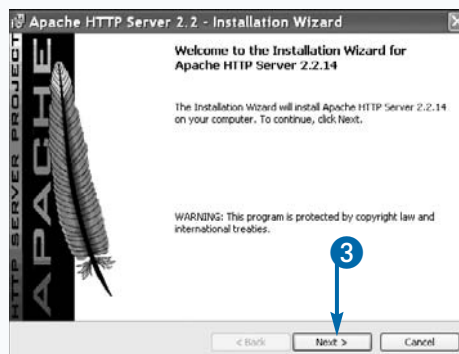
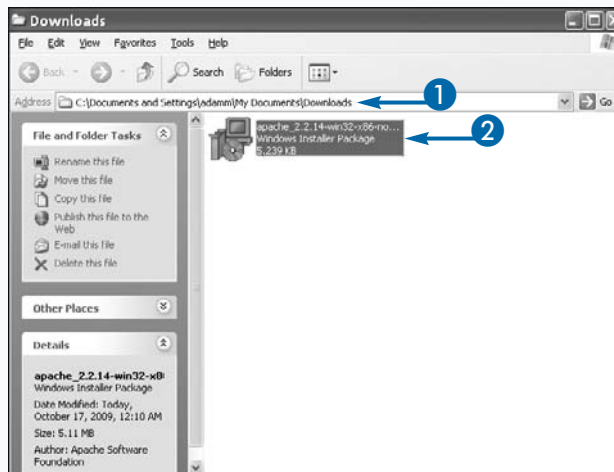
Install Apache for Windows

- 1 Open the folder containing the Apache download.
- 2 Double-click the Apache Installer.

The Apache HTTP Server Installation Wizard appears.

- 3 Click Next.
- 4 Click to select the I accept the terms in the license agreement option to accept the terms.
- 5 Click Next.
- 6 Scroll down and read the document, and then click Next.

The Apache Server readme document opens.



The Server Information screen appears.

7 Type your network domain name.

8 Type your server's name.

9 Type your e-mail address.

10 Select the for All Users option to install for all users.

11 Click Next.

The Setup Type screen appears.

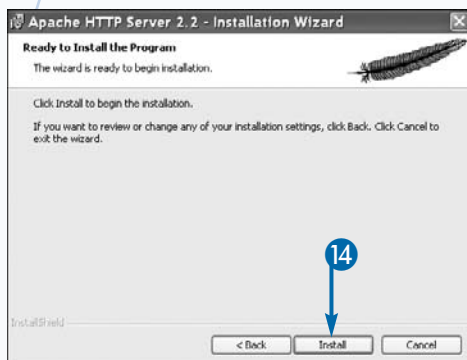
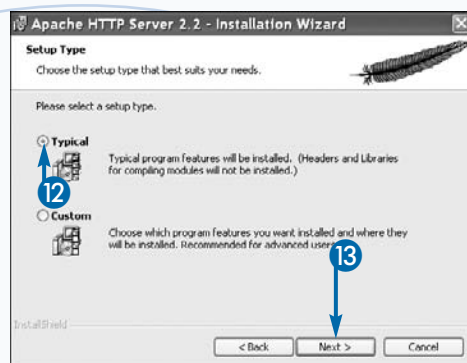
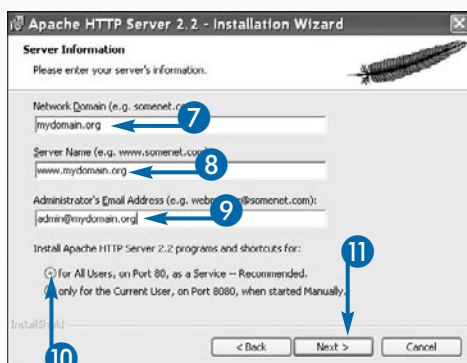
12 Click to select the Typical option.

13 Click Next.

The wizard is ready to begin the installation.

14 Click Install.

Apache for Windows is installed.



Extra

At installation time, you can safely ignore the options you populated on the wizard's Server Installation screen. This is useful if you do not know these values yet, as they will not have any impact on any test data or servers that are not yet receiving any live traffic. For setting these values on an Apache install after the installation is completed, see the section, "Configure Apache on Windows."

To confirm that the installation is complete, open a Web browser on the Windows server where you performed the installation. Using your browser, go to the address `http://localhost`. If you see the text, "It works!" in your browser, then the installation was a success.

Apache for Windows also sets up a Windows Service for itself. This automatically starts the Apache Web server each time you boot up your computer. When running as a service, you do not need to be logged in to a Windows machine in order to accept Web traffic.

An Apache icon appears in the Taskbar notification area, indicating the current status of the server. You can also use this icon to stop and restart the Apache Web service.

Configure Apache on Windows

The Apache configuration is handled by a central configuration file, `httpd.conf`, which contains hundreds of configuration directives that define how the Apache Web server functions on your computer. Within the configuration file, all comments begin with the hash (#) character. All text following a hash character to the end of the line is ignored. A configuration directive always begins with a specific command (exactly one word), followed by one or more values:

```
ServerAdmin admin@mydomain.org
```

The configuration file is divided into several different *Configuration Section Containers*, which define how and when a specific directive is applied. These containers apply directives only when a particular directory or file is read, when a particular external module is available, or when a particular host or URL is requested.

A Configuration Section Container looks like an HTML statement. It begins with a configuration directive within angle brackets, and ends with the same directive name preceded by a forward slash (/). In this example, the directive `DirectoryIndex` is only valid if the `dir_` module is available.

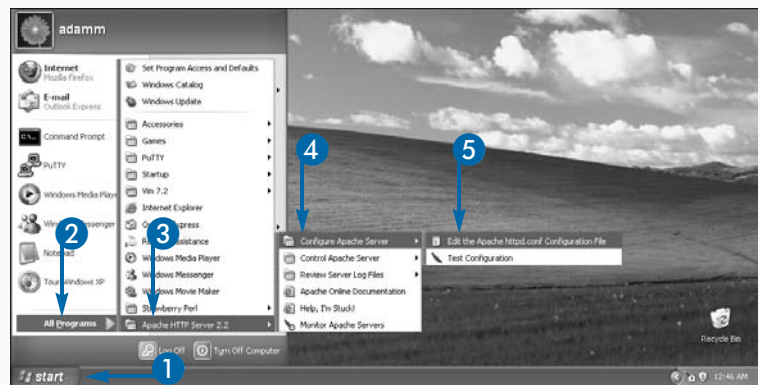
```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

Remember, if you make any modifications to the Apache configuration files in the Directory Index, you need to restart the Apache service before the changes take effect.

The full list of available directives, along with the reference manual, user guide, and several tutorials, are available on the Apache Documentation Web site at <http://httpd.apache.org/docs/2.2>.

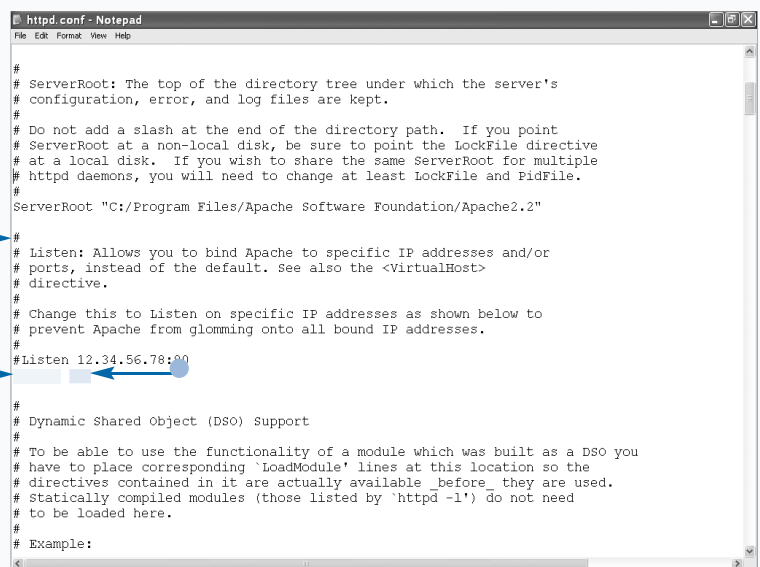
Configure Apache on Windows

- 1 Click Start.
- 2 Click All Programs.
- 3 Click Apache HTTP Server.
- 4 Click Configure Apache Server.
- 5 Click Edit the Apache `httpd.conf` Configuration File.



The `httpd.conf` file opens in a text editor.

- 6 Press the Page-Down key.
- Hash characters indicate comments in the configuration file.
 - A command is always the first word in a directive.
 - Additional values are used as arguments to the command directive.



7 Press the Page-Down key.

- A configuration section container begins.
- Directives within the container only apply when the container is true.
- The container section ends.

```

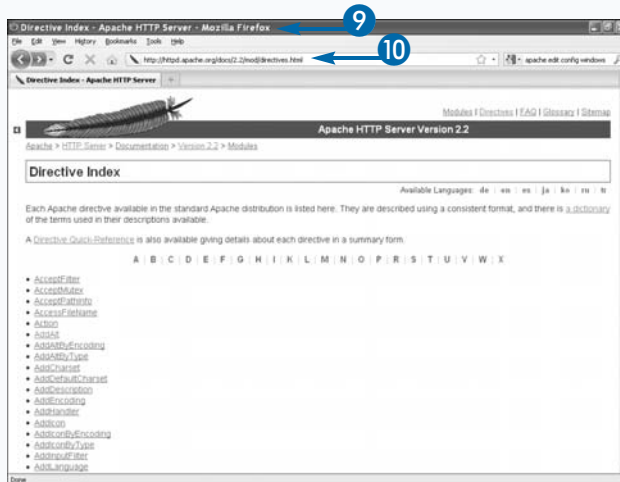
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs"
#
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs">

```

8 Open a Web browser.

9 Type <http://httpd.apache.org/docs/2.2/mod/directives.html> and press Enter.

The Directive Index appears in your browser.



Extra

Many Windows users may be intimidated by the depth of the configuration file, and the lack of a graphical interface to configure Apache. This is due in part to its Unix heritage, and many fans of Apache view this as a strength, not a weakness. If the Apache configuration were converted into a graphical user interface in Windows, it would be overly cluttered and crowded, trying to present too many sliders and buttons to fully convey what Apache has to offer.

There are some third-party utilities on the Internet that you can use to configure the `httpd.conf` file. The most popular is Apache-GUI, available from www.apache-gui.com.

Depending on the configuration directive you use, Apache allows for a command to be used within several context levels. A command may be allowed in the top-level server configuration, the `<virtualhost>` container, the `<directory>` container, or a special file called `.htaccess`. Directives applied at the `.htaccess` level perform just like the `<directory>` container, except that the `.htaccess` file should be saved within the actual directory. Also, `.htaccess` directives can *override* any preceding directives found in the other three contexts. For more information about the contexts, overrides, and other terms used to describe directives, go to <http://httpd.apache.org/docs/2.2/mod/directive-dict.html>.

Start and Stop the Apache Service on Windows

Because Apache runs as a service on Windows systems, it never appears as a traditional Windows application. Instead, it runs in the background, constantly listening for any incoming requests on your Internet connection.

The Apache service constantly monitors its own internal memory usage, and the number of active subprocesses, or *children*, actively waiting to fulfill a request. You can configure these settings in the Apache configuration file, `httpd.conf`.

If you change the configuration file, you need to restart the Apache service in order to apply the new configuration directives. It is important to note that restarting the server is only required with `httpd.conf` changes. Configuration directives applied to a specific directory, by way of an `.htaccess` file, are applied automatically. Changes to this file make restarting the service unnecessary.

Under normal conditions, the service promptly starts up as Windows boots, and gracefully stops when you shut down the system. You do not need to do this manually. It is only when the configuration file is altered that the service requires an Administrator to manually stop and restart the Apache service.

Any modifications to the CGI code or static HTML files do not require that you restart Apache.

When Apache is running on Windows, a tray icon does appear in the Windows Taskbar notification area; this allows an Administrator to quickly launch the Apache Monitor program. From here, you can review, stop, and restart the Apache service status. You can even use the Apache Monitor to connect remotely to another computer and monitor its Apache service status.

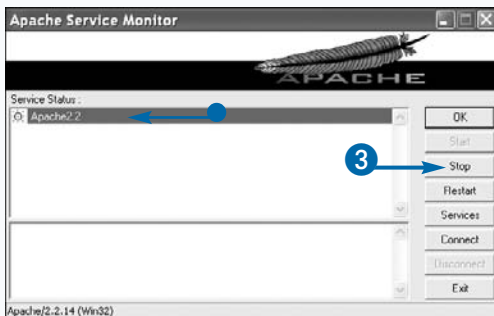
Start and Stop the Apache Service on Windows

- 1 Right-click the Apache Service Monitor icon (🔌) in the system tray.
- 2 Click Open Apache Monitor.




The Apache Service Monitor runs.

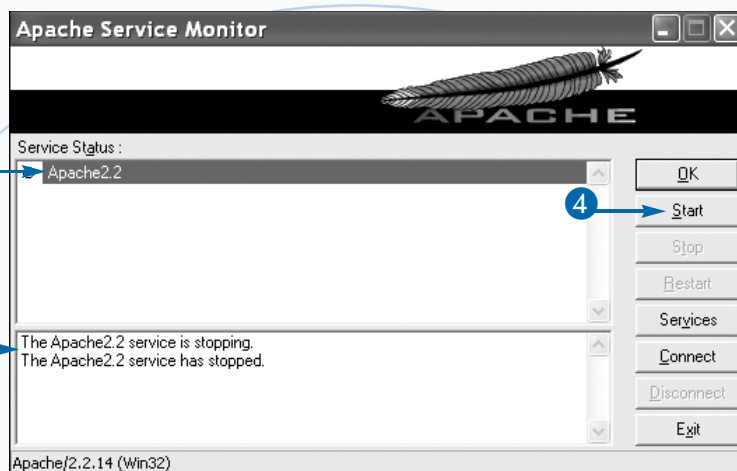
- The service status for Apache2.2 is green and running.
- 3 Click Stop.



- The Service Status icon turns red and stops.
- A running log of the Apache2.2 service appears.


The Apache Service Monitor icon in the system tray changes to a red square ().

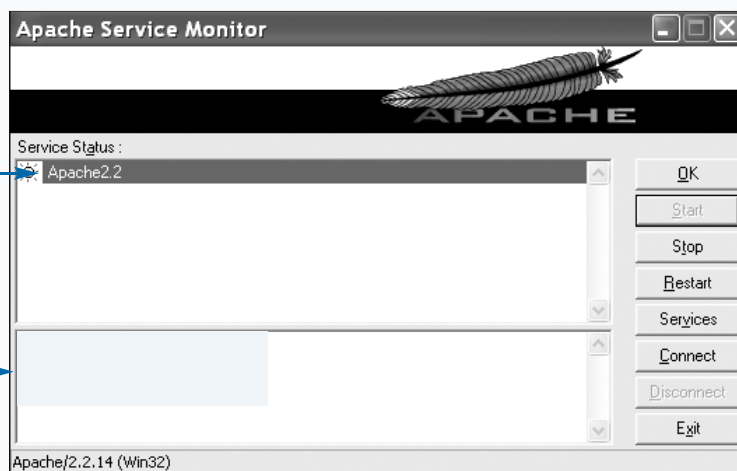
- 4 Click Start.



- The Service Status icon is green, indicating that it is running.

- The log has been appended.

The Apache Service Monitor icon in the system tray changes to a green arrow ().



Extra

Various alternatives are available in Windows to control the Apache service. Along with the example shown here, you can also use the Start Menu's Apache HTTP Server 2.2 program directory, the Service Control Panel, or even the command line.

To open the Service Control Panel, click Start → Control Panel → Administrative Tools → Services. Find the service named either "HTTP" or "HTTP SSL." To control the service on the command line, open a DOS prompt and run the commands, `net stop http` and `net start http`.

Before restarting the service, if an invalid configuration directive is written to the Apache configuration file, a problem occurs and the server fails to start back up due to a syntax failure. The Apache Server Monitor may not always display the error in its log window; however, you can validate the Apache configuration by running the Test Configuration utility.

You can run the utility by clicking the Start Menu's Apache HTTP Server 2.2 program directory, clicking Configure Apache Server, and then clicking Test Configuration. A window appears, explaining the invalid directive and its line number in `httpd.conf`. The invalid command also appears in the Apache `error.log` file. For more information on Apache logging, see Chapter 10.

Install Apache for Debian/Ubuntu Linux

Debian and Ubuntu systems use the DEB packaging format. You can use the suite Advanced Package Tool, or APT, for downloading, installing, and removing DEB packages. On Debian- and Ubuntu-based Linux systems, the Apache Web server DEB package and program name are often identified as `apache2`.

The installation process requires that your Linux computer be connected to the Internet. The download, installation, and setup process is handled by a single command, `apt-get`, which you should execute in a Terminal window.

It is beneficial to run a local copy of the Apache Web server, even if a router or firewall separates your connection to the Internet. You can use a local copy for testing your own CGI code before putting it online.

Extra

The program `apt-get` is not the only way to install a package on a Debian- or Ubuntu-based system. Alternatively, you can use `aptitude` with the command `sudo aptitude install package`, or, if you prefer a graphical interface, you can use the Synaptic Package Manager.

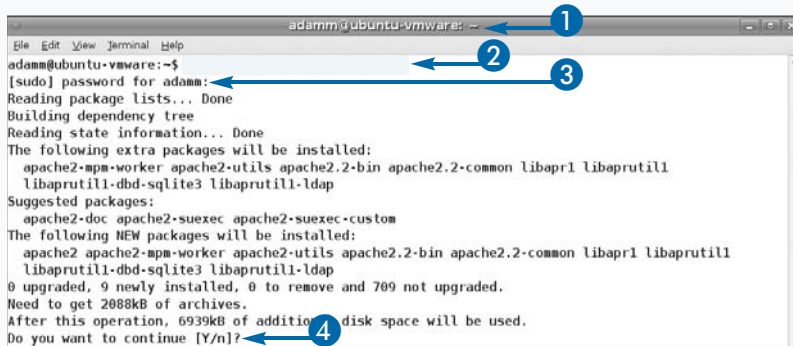
One of the packages suggested by `apt-get` is `apache2-doc`. This is the documentation package for the Apache 2.2 server. While this information is available online, having it local on the system can be useful. Once you install it, you can access it from `http://localhost/manual/`.

Install Apache for Debian/Ubuntu Linux

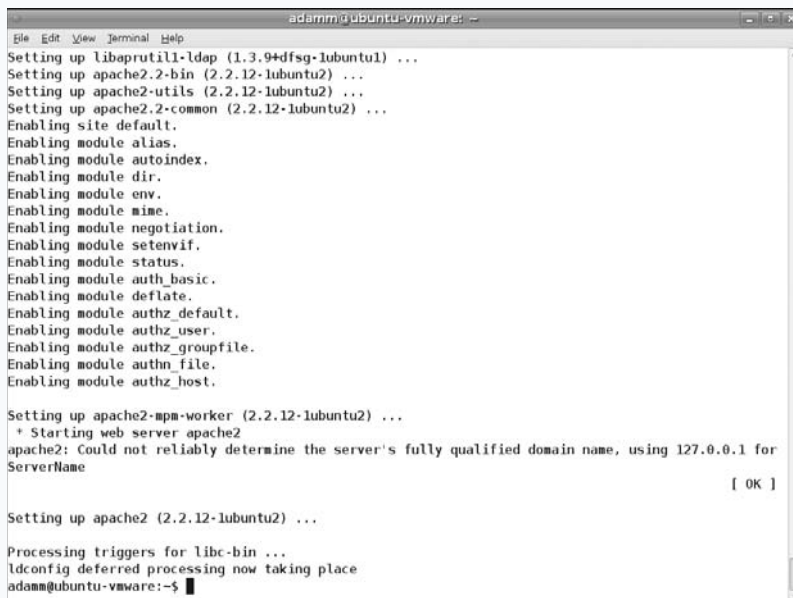
- 1 Open a Terminal window on Debian or Ubuntu.
- 2 Type `sudo apt-get install apache2` and press Enter.
- 3 Type your password if prompted.
- 4 Type `Y` and press Enter to continue.

The Apache Server downloads and installs.

Note: If Apache is already installed, this process also upgrades it, if an upgrade is available online.



```
adam@ubuntu-vmware: ~$ sudo apt-get install apache2
[sudo] password for adam:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap
Suggested packages:
  apache2-doc apache2-suexec apache2-suexec-custom
The following NEW packages will be installed:
  apache2 apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap
0 upgraded, 9 newly installed, 0 to remove and 709 not upgraded.
Need to get 2088kB of archives.
After this operation, 6939kB of additional disk space will be used.
Do you want to continue [Y/n]?
```



```
Setting up libaprutil1-ldap (1.3.9+dfsg-1ubuntu1) ...
Setting up apache2.2-bin (2.2.12-1ubuntu2) ...
Setting up apache2-utils (2.2.12-1ubuntu2) ...
Setting up apache2.2-common (2.2.12-1ubuntu2) ...
Enabling site default.
Enabling module alias.
Enabling module autoindex.
Enabling module dir.
Enabling module env.
Enabling module mime.
Enabling module negotiation.
Enabling module setenvif.
Enabling module status.
Enabling module auth_basic.
Enabling module deflate.
Enabling module authz_default.
Enabling module authz_user.
Enabling module authz_groupfile.
Enabling module authn_file.
Enabling module authz_host.

Setting up apache2-mpm-worker (2.2.12-1ubuntu2) ...
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1 for
ServerName

Setting up apache2 (2.2.12-1ubuntu2) ...

Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
adam@ubuntu-vmware: ~$
```

Install Apache for Red Hat Linux

On most Linux servers, Apache should already be installed as a standard component. All Red Hat-based Linux distributions distribute software packages using the RPM packaging format. You can use the program suite Yellowdog Updater, Modified (YUM) for downloading, installing, and removing RPM packages. On Red Hat-based Linux systems, the Apache Web server RPM package and program name are often identified simply as “httpd”.

YUM is not the only way to install a package on a Red Hat-based system. Graphical workstations can use the KPackage if you use the KDE window manager, or GnoRPM if you use the Gnome window manager. Check your particular Linux Distribution's documentation and help manual for information on

how to use an alternative graphical program. Fortunately, all Red Hat-based systems do support YUM on the command line. The installation process requires that your computer be connected to the Internet. The download, installation, and setup process is handled by a single command, `yum`, which you should execute in a Terminal window.

It is beneficial to run a local copy of the Apache Web server, even if a router or firewall separates your connection to the Internet. You can use a local copy for testing your own CGI code before putting it online.

You can always make HTTP requests to your own server using the special hostname, *localhost*. This is actually an alias to the IP address 127.0.0.1, which always acts as the address to the computer you are actively working on. Once you install Apache, you can access all programs and files in a browser using the alias `http://localhost`.

Install Apache for Red Hat Linux

- 1 Open a Terminal window on a Red Hat system.
- 2 Type `su -c 'yum install httpd'` and press Enter.
- 3 Type root's password.

Note: If `sudo` is available on your system, type `sudo yum install httpd` instead. Then type your own password.

- 4 Type `Y` and press Enter to continue.

- The Apache RPM package downloads and installs.

Note: If Apache is already installed, this process also upgrades it, if an upgrade is available online.

```
centos@localhost:~# su -c 'yum install httpd'
Password:
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.telus.net
 * updates: ftp.telus.net
 * addons: ftp.telus.net
 * extras: ftp.telus.net
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
--> Package httpd-2.2.3-22.el5.centos.2 set to be updated
--> Processing Dependency: httpd = 2.2.3-22.el5.centos for package: mod_ssl
--> Running transaction check
--> Package mod_ssl-1.0.2-22.el5.centos.2 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch      Version              Repository           Size
=====
Updating:
httpd        i386      2.2.3-22.el5.centos.2 updates             1.2 M
mod_ssl      i386      1.0.2-22.el5.centos.2 updates              87 k
=====

Transaction Summary
=====
Install      0 Package(s)
Update      2 Package(s)
Remove      0 Package(s)
=====
Total download size: 1.3 M
Is this ok [y/N]: Y
```

```
centos@localhost:~# su -c 'yum install httpd'
Password:
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.telus.net
 * updates: ftp.telus.net
 * addons: ftp.telus.net
 * extras: ftp.telus.net
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
--> Package httpd-2.2.3-22.el5.centos.2 set to be updated
--> Processing Dependency: httpd = 2.2.3-22.el5.centos for package: mod_ssl
--> Running transaction check
--> Package mod_ssl-1.0.2-22.el5.centos.2 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch      Version              Repository           Size
=====
Updating:
httpd        i386      2.2.3-22.el5.centos.2 updates             1.2 M
mod_ssl      i386      1.0.2-22.el5.centos.2 updates              87 k
=====

Transaction Summary
=====
Install      0 Package(s)
Update      2 Package(s)
Remove      0 Package(s)
=====
Total download size: 1.3 M
Is this ok [y/N]: Y
Downloading Packages:
(1/2): mod_ssl-2.2.3-22.el5.centos.2.1386.rpm | 87 kB 00:00
(2/2): httpd-2.2.3-22.el5.centos.2.1386.rpm | 1.2 MB 00:00
Total 1.0 MB/s | 1.3 MB 00:01
warning: rpm: hdrFromFds: Header V3 DSA signature: NOKEY, key ID e8562897
Importing GPG key 0x08562897 "CentOS-5 Key (CentOS 5 Official Signing Key) <centos-5-key@centos.org>" from /etc
/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
Is this ok [y/N]: Y
Running rpm check debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating      : httpd                  [1/4]
  Updating      : mod_ssl                [2/4]
  Cleanup       : httpd                  [3/4]
  Cleanup       : mod_ssl                [4/4]
Updated: httpd-2.2.3-22.el5.centos.2 mod_ssl-1.0.2-22.el5.centos.2
Complete!
centos@localhost:~#
```

Configure Apache on Linux

The Apache configuration is handled by a central configuration directory, `/etc/apache2` on Debian/Ubuntu, or `/etc/httpd` on Red Hat. Within this directory, the first file read is the global configuration file, `apache2.conf` on Debian/Ubuntu, or `conf/httpd.conf` on Red Hat. Next is the subdirectories, `conf.d`, `mods-enabled`, and `sites-enabled`, if they exist.

Within each configuration file read, the same syntax applies. All comments begin with the hash (`#`) character. All text following a hash character to the end of the line is ignored. A configuration directive always begins with a specific command, followed by one or more values.

```
ServerAdmin admin@mydomain.org
```

The configuration file is divided into several different *Configuration Section Containers*, which define how and

when a specific directive is applied. These containers apply directives only when a particular directory or file is read, when a particular external module is available, or when a particular host or URL is requested.

A Configuration Section Container looks like an HTML statement. It begins with a configuration directive within angle brackets, and ends with the same directive name preceded by a forward-slash (`/`). In this example, the directive `DirectoryIndex` is only valid if the `dir_` module module is available.

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

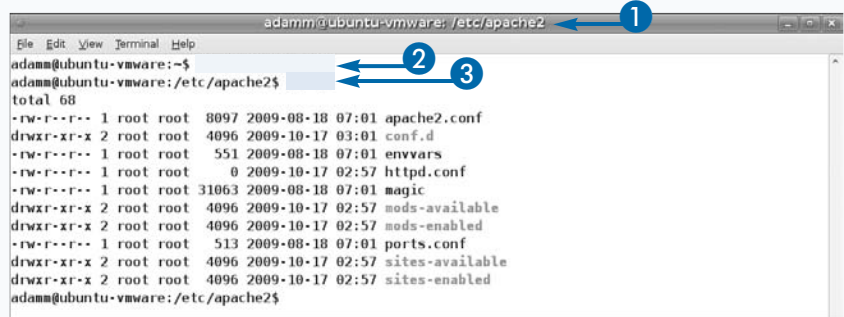
If you make any modifications to the Apache configuration files in the directory index, restart the Apache daemon.

Configure Apache on Linux

- 1 Open a Terminal window.
- 2 Type `cd /etc/apache2/` (use `/etc/httpd/conf` on Red Hat).
- 3 Type `ls -l` to list the directory contents.
- 4 Open the file `apache2.conf` (open `httpd2.conf` on Red Hat) in a text editor.

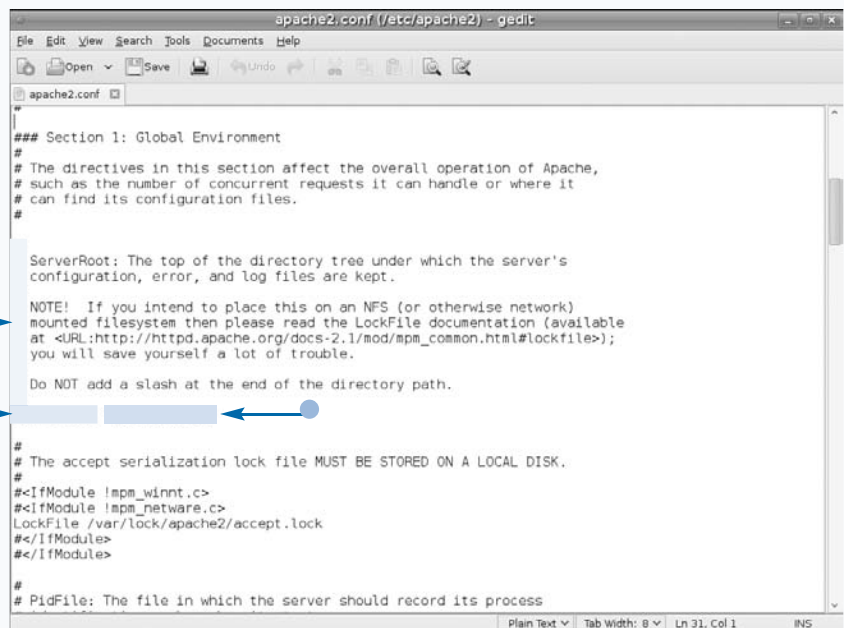
Note: On Gnome systems, type *gedit apache2.conf* to open an editor.

- The `apache2.conf` file opens in a text editor.
- 5 Press the Page Down key.
- Hash characters indicate comments in the configuration file.
 - A command is always the first word in a directive.
 - Additional values are used as arguments to the command directive.



A terminal window titled 'adamm@ubuntu-vmware: /etc/apache2' with a blue arrow pointing to the title bar labeled '1'. The prompt is 'adamm@ubuntu-vmware:~\$'. The user enters 'cd /etc/apache2/' and the prompt changes to 'adamm@ubuntu-vmware:/etc/apache2\$'. The user enters 'ls -l' and the output is listed. A blue arrow points to the prompt area labeled '2', and another blue arrow points to the output area labeled '3'.

```
adamm@ubuntu-vmware:~$ cd /etc/apache2/
adamm@ubuntu-vmware:/etc/apache2$ ls -l
total 68
-rw-r--r-- 1 root root 8097 2009-08-18 07:01 apache2.conf
drwxr-xr-x 2 root root 4096 2009-10-17 03:01 conf.d
-rw-r--r-- 1 root root 551 2009-08-18 07:01 envvars
-rw-r--r-- 1 root root 0 2009-10-17 02:57 httpd.conf
-rw-r--r-- 1 root root 31063 2009-08-18 07:01 magic
drwxr-xr-x 2 root root 4096 2009-10-17 02:57 mods-available
drwxr-xr-x 2 root root 4096 2009-10-17 02:57 mods-enabled
-rw-r--r-- 1 root root 513 2009-08-18 07:01 ports.conf
drwxr-xr-x 2 root root 4096 2009-10-17 02:57 sites-available
drwxr-xr-x 2 root root 4096 2009-10-17 02:57 sites-enabled
adamm@ubuntu-vmware:/etc/apache2$
```



A Gedit text editor window titled 'apache2.conf (/etc/apache2) - gedit'. The file 'apache2.conf' is open. The content shows configuration directives and comments. Blue arrows point to specific parts of the file: one points to a comment line starting with 'NOTE!', another points to a comment line starting with 'Do NOT', and a third points to a comment line starting with 'The accept serialization lock file MUST BE STORED ON A LOCAL DISK.'.

```
### Section 1: Global Environment
#
# The directives in this section affect the overall operation of Apache,
# such as the number of concurrent requests it can handle or where it
# can find its configuration files.
#
ServerRoot: The top of the directory tree under which the server's
configuration, error, and log files are kept.

NOTE! If you intend to place this on an NFS (or otherwise network)
mounted filesystem then please read the LockFile documentation (available
at <URL:http://httpd.apache.org/docs-2.1/mod/mpm_common.html#lockfile>;
you will save yourself a lot of trouble.

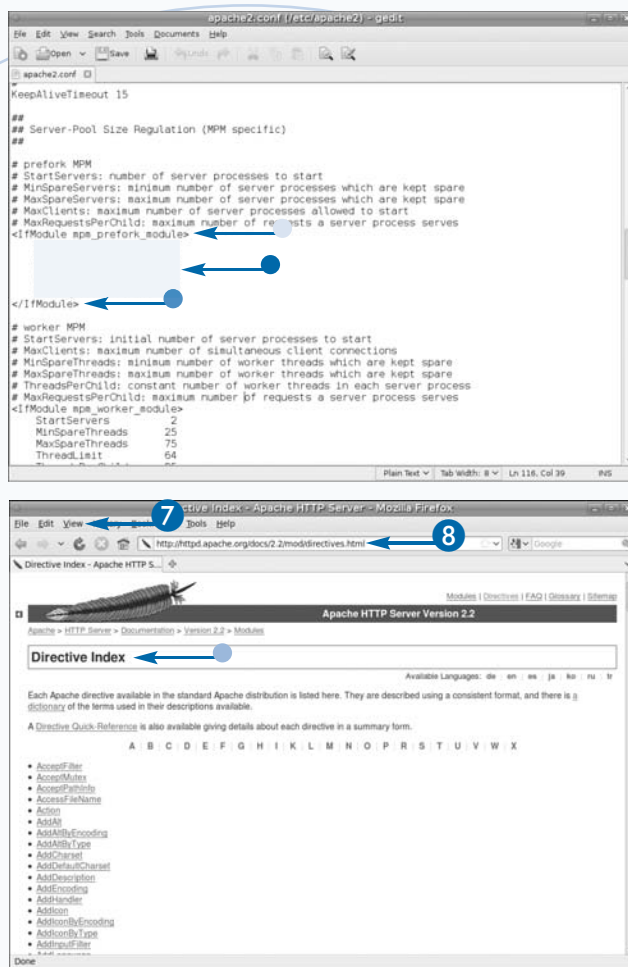
Do NOT add a slash at the end of the directory path.

#
# The accept serialization lock file MUST BE STORED ON A LOCAL DISK.
#
#<IfModule !mpm_winnt.c>
#<IfModule !mpm_network.c>
LockFile /var/lock/apache2/accept.lock
#</IfModule>
#</IfModule>

#
# PidFile: The file in which the server should record its process
```

- 6 Press the Page Down key.
 - A configuration section container begins.
 - Directives within the container only apply when the container is true.
 - The container section ends.
- 7 Open a Web browser.
- 8 Go to the address <http://httpd.apache.org/docs/2.2/mod/directives.html>.
 - The directive index loads.

Note: The full list of available directives, along with the reference manual, user guide, and several tutorials, are available on the Apache Documentation Web site at <http://httpd.apache.org/docs/2.2>.



Extra

The Apache online documentation provides a complete listing and description of all available configuration directives. You can access this information by going to <http://httpd.apache.org/docs/2.2/mod/directives.html> in your Web browser.

Depending on the configuration directive used, Apache enables you to use a command within several context levels. A command may be allowed in the top-level server configuration, the `<virtualhost>` container, the `<directory>` container, or a special file called `.htaccess`. Directives applied at the `.htaccess` level perform just like the `<directory>` container, except that you should save the `.htaccess` file within the actual directory. Also, `.htaccess` directives can overrule any preceding directives found in the other three contexts.

For more information about the contexts, overrides, and other terms used to describe directives, go to <http://httpd.apache.org/docs/2.2/mod/directive-dict.html>.

Start and Stop the Apache Service on Linux

Because Apache runs as a daemon on Linux systems, it never appears as a traditional application. Instead, it runs in the background, constantly listening for any incoming requests on your Internet connection.

The Apache service constantly monitors its own internal memory usage, and the number of active subprocesses, or children, actively waiting to fulfill a request. You can configure these settings in the Apache configuration file, `httpd.conf`.

If you change the configuration file, you need to restart the Apache service in order to apply the new configuration directives. It is important to note that you only need to restart the server if any configuration files change. Configuration directives applied to a specific directory, by way of an `.htaccess` file, are applied automatically. Changes to these files make restarting the service unnecessary.

Under normal conditions, the service promptly starts up as Linux boots, and gracefully stops when you shut down the system. You do not need to do this manually. It is only when a configuration file is altered that the service requires an Administrator to manually stop and restart the Apache service.

Any modifications to the CGI code or static HTML files do not require you to restart Apache.

Some command-line programs exist for manipulating the Apache server's status; the most commonly used ones are `apache2ctl` on Debian/Ubuntu, and `apachectl` on Red Hat.

You need to be careful if you frequently restart a production Web server that is serving live traffic. You can affect downloads or Web-page requests if you restart the server in mid-transfer. Instead, use the Apache `graceful` restart command.

Start and Stop Apache Service on Linux

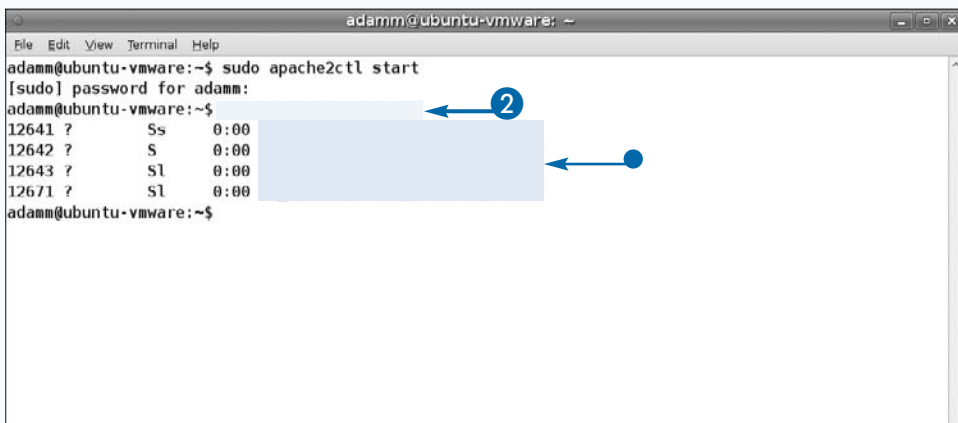
- 1 Type `sudo apache2ctl start` (use `apachectl` on Red Hat) and press Enter.

Note: If you do not find `apache2ctl`, you may need to provide its full directory path, `/usr/sbin/apache2ctl`.



```
adamm@ubuntu-vmware: ~  
File Edit View Terminal Help  
adamm@ubuntu-vmware:~$ sudo apache2ctl start  
[sudo] password for adamm:  
adamm@ubuntu-vmware:~$
```

- 2 Type `ps fax | grep apache2` and press Enter.
 - Apache is running.



```
adamm@ubuntu-vmware: ~  
File Edit View Terminal Help  
adamm@ubuntu-vmware:~$ ps fax | grep apache2  
[sudo] password for adamm:  
adamm@ubuntu-vmware:~$  
12641 ?      Ss      0:00  
12642 ?      S       0:00  
12643 ?      Sl      0:00  
12671 ?      Sl      0:00  
adamm@ubuntu-vmware:~$
```

- 3 Type **sudo apache2ctl stop**.
- The Apache process list returns nothing.

```

adammm@ubuntu-vmware: ~
File Edit View Terminal Help
adammm@ubuntu-vmware:~$ sudo apache2ctl start
[sudo] password for adammm:
adammm@ubuntu-vmware:~$ ps fax | grep apache2
12641 ?      Ss   0:00 /usr/sbin/apache2 -k start
12642 ?      S    0:00 \_ /usr/sbin/apache2 -k start
12643 ?      Sl   0:00 \_ /usr/sbin/apache2 -k start
12671 ?      Sl   0:00 \_ /usr/sbin/apache2 -k start
adammm@ubuntu-vmware:~$
adammm@ubuntu-vmware:~$ ps fax | grep apache2
adammm@ubuntu-vmware:~$

```

- 4 Type **sudo apache2ctl restart**.
- If Apache is not running when restart is called, a message displays.
 - Apache is running.

Note: The *restart* command simply stops and starts the Apache service.

```

adammm@ubuntu-vmware: ~
File Edit View Terminal Help
adammm@ubuntu-vmware:~$ sudo apache2ctl start
[sudo] password for adammm:
adammm@ubuntu-vmware:~$ ps fax | grep apache2
12641 ?      Ss   0:00 /usr/sbin/apache2 -k start
12642 ?      S    0:00 \_ /usr/sbin/apache2 -k start
12643 ?      Sl   0:00 \_ /usr/sbin/apache2 -k start
12671 ?      Sl   0:00 \_ /usr/sbin/apache2 -k start
adammm@ubuntu-vmware:~$ sudo apache2ctl stop
adammm@ubuntu-vmware:~$ ps fax | grep apache2
adammm@ubuntu-vmware:~$
httpd not running, trying to start
adammm@ubuntu-vmware:~$ ps fax | grep apache2
12735 ?      Ss   0:00
12736 ?      S    0:00
12737 ?      Sl   0:00
12761 ?      Sl   0:00
adammm@ubuntu-vmware:~$

```

Apply It

To mitigate the effect on downloads or Web-page requests when restarting the Apache Web server, Apache provides the `graceful` command to `apache2ctl`. A graceful restart waits for all active requests to finish before the server is restarted.

TYPE THIS

```
sudo apache2ctl graceful
```



RESULTS

Apache is gracefully restarted after all data requests have been processed.

If you restart Apache with a bad configuration, it does not come back up until the configuration error has been resolved. Apache does provide a test as the `configtest` command to `apache2ctl`. This tests the current configuration directives and validates whether the server can be brought back online after shutdown.

TYPE THIS

```
sudo apache2ctl configtest
```

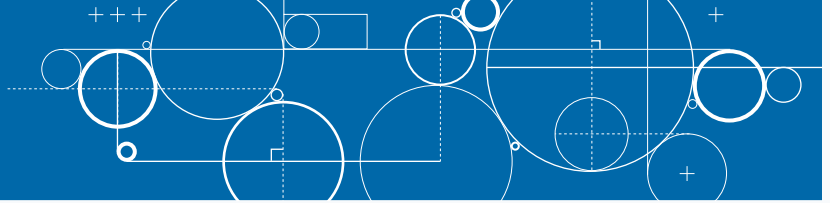


RESULTS

Syntax OK

Note that the `graceful restart` command automatically runs `configtest`. If it fails, the system does not shut down.

Understanding Perl Syntax



Perl is very lenient when it comes to its own formatting. If programmers have their own

personal style of formatting, Perl readily conforms to it, provided they follow some basic syntax rules.

Capitalization

All Perl functions, variables, modules, and subroutines are case-sensitive. All built-in functions and internal modules use lowercase names. All user-defined variables, modules, and subroutines may contain any mix of upper- and lowercase characters. Only user-defined modules should begin with an uppercase character.

Line Spacing and Formatting

Perl is very relaxed when it comes to line spacing and formatting, leaving it up to the programmer's personal preference. Multiple lines and spaces do not affect execution, unless they are contained within single- or double-quotes, in which case they are treated as literal spacing. The Perl Style Guide contains a list of recommended guidelines. You can access the guide in the Perl Documentation program by using the command, `perldoc perlstyle`.

Semicolons

You use the semicolon (;) to indicate the end of a command in Perl. Usually, there is one command per line of code, making it look like every line needs a semicolon. As the Perl interpreter reads the file, it executes the

preceding command each time it reads a semicolon. It is legal for a command to span multiple lines and spacing, provided that it ends in a semicolon. Similarly, it is possible to have multiple commands (and semicolons) per line.

Brackets

As with written language, you can use bracket pairs to group multiple objects together. Brackets also provide a form of hierarchical context between objects that are included and excluded.

Parentheses

Parentheses (round brackets) are often preceded by functions and commands to specify arguments. While not required, they are a good code practice as they clearly separate commands from arguments. For example, Perl interprets the following two lines of code in the same way.

```
&MyCommand( 1, 2, "abc" );
&MyCommand 1, 2, "abc";
```

You can also use parentheses in mathematical operations.

```
print 5*2+5;      # returns 15
print 5*(2+5);    # returns 35
```

Square Brackets

You can use square brackets to retrieve individual values from arrays.

```
@foo = ( 'abc', 'def', 'ghi' );
print $foo[ 1 ];      # returns 'def'
```

Curly Brackets

You can use curly brackets to retrieve individual values from hashes.

```
%foo = ( 'abc' => 123,
        'def' => 456,
        'ghi' => 789 );
print $foo{ 'abc' };    # returns '123'
```

You can also use curly brackets to define blocks of code that are only executed if a conditional statement is true, or a subroutine is executed.

```
if ( $foo{ 'abc' } > 100 ) {
    print "Greater than 100!\n";
}
```

Angle Brackets

You normally use angle brackets in relational operations, or in binary shift operations. A third use of the angle bracket is as the Arrow operator (represented as `->`). Arrows are often used as a dereference point for accessing variables and subroutines supplied by an external module or referenced variable.

Comments

You use comments for writing text in a Perl script that will be ignored by the Perl interpreter. As a result, you can include additional documentation by code that may otherwise appear questionable. You use the hash sign (#) to prefix all comments. When it appears, the Perl interpreter ignores all text following the hash sign until the end of the line. It is legal for the hash sign to appear as the first character in the line, in which case the Perl interpreter ignores the entire line.

Relational Operations

You use relational operators for comparing two values together within an `if` or `elsif` conditional statement.

CHARACTER(S)	COMMAND	DESCRIPTION
<	lt	Less-than
<=	le	Less-than or equal-to
>	gt	Greater-than
>=	ge	Greater-than or equal-to
==	eq	Equal-to
!=	ne	Not-equal-to

Logic Operations

Logic operations are normally found within `if` or `elsif` commands as a way to string multiple comparison operations.

CHARACTER(S)	COMMAND	DESCRIPTION
&&	AND	Both sides of the AND operation must be true for the equation to be true.
	OR	Either side of the OR operation may be true for the equation to be true.
!	NOT	This command reverses a true operation to be false, and a false operation to be true.

Special Characters

Special characters in Perl are always preceded by a single backslash (\). You can reproduce ASCII characters using the backslash followed by an octal or hexadecimal code. Octal ASCII codes are represented by a backslash and three numbers (for example, `\156`), while hexadecimal ASCII codes contain a backslash, an 'x', and two hexadecimal digits (for example, `\x6E`). (Both of these examples describe the letter *n*.)

Carriage Return and Line Feed Characters

The “new line” characters are the carriage return (represented as `\r`) and the line feed (represented as `\n`). They have a particular meaning, depending on the context and the operating system you are working on. When reading text files into Perl, it is important to pay attention to the format of the file when parsing each line.

OPERATING SYSTEM	CHARACTER NAME(S)	CHARACTER(S)
Windows	CRLF	<code>\r\n</code>
Unix	LF	<code>\n</code>
Mac OS	CR	<code>\r</code>

When printing content to the user (or writing content to a file), regardless of the operating system, Perl correctly identifies the “new line” character being written when the developer simply uses a line feed (`\n`). For example, the

following code works correctly on all three operating systems.

```
print "Hello, world\n";
```

Tab Character

You can use the tab character (represented as `\t`) to align printed text, just like a word-processor. Every tab character shifts the cursor eight spaces to the right. To use a different tab stop, you can use the module `Text::Wrap` and modify `$Text::Wrap::tabstop`.

Backslash Character

The literal backslash character (represented as `\\`) is useful when you need a single backslash. You can also use a backslash character to reference another variable (represented as `\variable`) and this provides a pointer to that variable, similar to C and C++.

Understanding Perl Syntax (continued)

Variables

There are three main types of variables in Perl: scalars, arrays, and hashes. You define a variable using a special character that indicates its type (\$, @, and %, respectively), followed by a name. A *scalar* is a variable that contains a single character, string, or object. An *array* is a series of variables in a set order. A *hash* is a group of indexed variables. It is possible to nest hashes and arrays to create complex data structures.

User-Defined Variables

You can create user-defined variables with the `my` command. The enclosing block of code determines the variable's *scope*. The scope dictates where a variable can be used and will retain its value. Locally-scoped variables only exist in the code block where they are declared; globally-scoped variables can be used anywhere as they are declared outside of any code blocks. You can redefine an existing variable in a new sub-block, but their original content is restored at the end of the sub-block.

You declare user-defined variables with the `my` command. Once you declare a variable, you can use it to store and retrieve information from anywhere within the block of code where the `my` command was called.

Predefined Variables

Some variable names are reserved by Perl and have a hidden meaning, often following a special nomenclature in that they appear to be defined out of nowhere. You can access a complete list of predefined variables in the Perl Documentation by using the command, `perldoc perlvar`.

Quotes

Perl utilizes both types of quoting characters, the double-quote (") and the single-quote ('), when setting variables or printing content. Both are useful around literal strings; however, a double-quote allows expansion of variables and special characters in the string; a single quote does not.

The following examples look identical, but the first resolves the scalar and new-line character. The second prints the literal text.

```
print "Hello, $user\n";
print 'Hello, $user\n';
```

The backquote (`) has a special meaning and should not be confused with a left-handed single-quote. A backquoted string actually launches a new command shell within your Perl program, executes the string as a command in real-time, and returns the output back to your program. For example:

```
my $currentDate = `date`;
```

Numerical content does not require quotes.

```
my $timeout = 5;
```

Built-In Functions

You can access every built-in Perl function by using the function name. Functions typically accept a series of variable arguments and return either an error code (useful for conditional logic) or a stand-alone scalar variable.

You can find a full list of available Perl functions in the Perl Documentation by using the command `perldoc perlfunc`. You can quickly reference a specific Perl function in the Perl Documentation by using the command `perldoc -f function`.

User-Defined Subroutines

User-defined subroutines act just like built-in functions in Perl, except that they must be defined within the script or supporting modules. You should create a subroutine when you identify an opportunity to use common code in multiple locations in the program.

Subroutines begin by accepting zero or more arguments (retrieved as scalars through the special `@_` array) and end by returning zero or more results (delivered to the caller with the `return` function). You can find more information about subroutines in the Perl Documentation by using the command `perldoc perlsub`.

Understanding the Anatomy of a Perl Script

The basic anatomy of a Perl script is relatively simple. An individual script file typically houses the core of the program, which may in

turn import external third-party modules, or user-defined common libraries.

Script File

The Perl interpreter accepts code in one of two ways: in a script file or on the command line. The command-line interface is not used with CGI development, so you need to save all of your code as files with the `.pl` extension.

Operating System-Specific Files

Before you can execute any code, you must associate Windows Perl scripts to the Perl interpreter binary using their `.pl` extension; Unix Perl scripts must have the execute-bit set and reference the Perl binary through their header.

Parameters

You can pass optional command-line parameters into any Perl script when you run them within a DOS Prompt or Terminal window. Command-line arguments are delivered using the `@ARGV` built-in array, and are available globally in the script.

Script Content

The Perl interpreter parses and executes the script in a linear fashion, from top to bottom. Modules allow for common code to be referenced from external files.

Header

The first line of all Perl scripts should be the path to the Perl executable. The header you use must represent the path specific to your operating system. If you are using a Unix system, the header is almost always this:

```
#!/usr/bin/perl
```

Because Windows Perl scripts are associated through the `.pl` extension file type, they do not require a header when they are executed on the command-line. However, when you are ready to executing Perl CGI scripts using Windows, Apache still requires a correct path:

```
#!C:/Perl/bin/perl.exe
```

The header is also a useful place to insert some introductory comments about the program, identifying the author, purpose, version, and copyright.

Content

Perl code written after the header is treated as globally accessible to your script. This includes all variables, modules, and subroutines that you have not defined within existing module and subroutine blocks. Content that you write inside modules and subroutines is treated as locally accessible only to that particular code block.

Reference External Code

If your script requires any external modules or common code, you must import it immediately after the header. You can use the `use`, `do`, or `require` commands to import that code into your script. Immediately after you have imported a module, some modules require a global scalar to act as a reference handle to the module's code. These handles may be defined anywhere after its module has been loaded.

User-Defined Subroutines and Built-In Functions

The Perl interpreter typically provides built-in functions. (User-defined subroutines and built-in functions are collectively referred to as *functions*.) All functions accept a single list of scalars as parameters, and return a similar list of scalars as results. (Because an array is actually a series of scalars, functions accept an anonymous array as input, and another array as output.)

Parameters passed into a function are accessible in the built-in array, `@_`. Once the function has completed its task, you use a `return` statement to exit it, supplying one or more scalars (or an array) back to the caller.

User-defined subroutines may exist anywhere in the code. Unlike with C or C++, you do not need to predefine subroutines. You can redefine subroutines at any time, overruling (or overloading) a previous definition.

Create a New Perl Script

You need very few tools to create a new Perl script. Specifically, you should have some experience with both the Terminal window and text editors, as well as be familiar with the location of the Perl binary, when developing Perl in a new environment. You do not need to use the Terminal to program Perl, but getting into the habit now will be helpful later, especially when validating script output and checking for errors. On most Unix systems, the programs Xterm and Gnome Terminal provide easy access to the command line.

The Terminal is also useful for quickly jumping between editing and executing a program by leaving the two windows open. By leaving your active program open in an editor, you can simply save the file when you are ready to test it, and then switch to the command line. By leaving the Terminal window open in the directory where

your program is saved, you can quickly execute the program with the command `perl program`. After executing the command once, you can save some time by pressing the up arrow to bring up the last command.

On most systems, you can find a text-editing program by navigating through the list of installed programs. On Windows, Notepad is available. On Unix, gedit or kwrite are great starting points. You can launch these programs from the command line with a filename as a parameter.

Locating the Perl binary is relatively easy. On Unix systems, using the command `which perl` typically yields `/usr/bin/perl`. On Windows, ActiveState Perl installs its binary into `C:\Perl\bin\perl.exe` by default. You need the location of the Perl binary for the header line of the script. You prefix the header with `#!`, followed by the path to the binary.

Create a New Perl Script

- 1 Open your Terminal window.

Note: You can open the Windows Terminal by clicking **Start** → **Run**. Type **cmd.exe** in the Run dialog box and press **Enter**.

- 2 In the Terminal window, start your preferred text editor with a new script, ending with the `.pl` extension.

Note: In Windows, if you do not have an editor installed, type **notepad.exe** “FILENAME.pl” and press **Enter**.

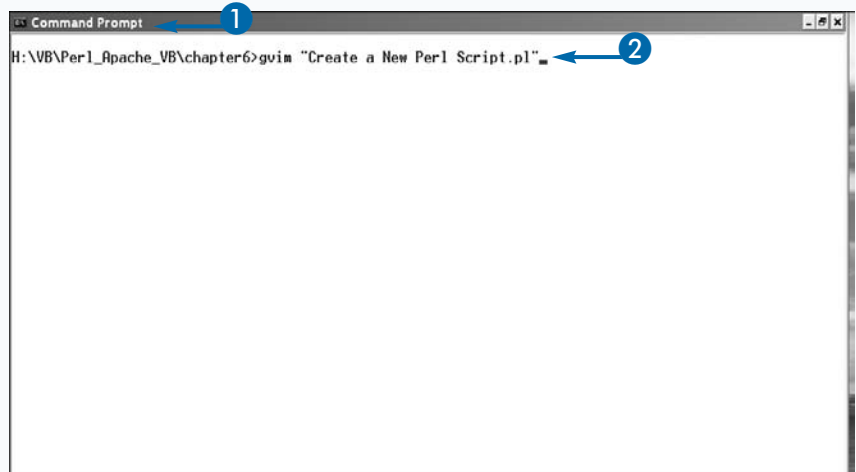
Note: Notepad asks for confirmation in order to create a new file. Click **Yes** to create it.

A blank text editor window opens.

- 3 Type `#!/usr/bin/perl` as the Perl header in the first line and press **Enter**.

Note: This is largely beneficial for Unix-only systems. On Windows, it has no effect. The actual location of the Perl binary may differ on your system.

- 4 Type `#` followed by your comment text, and then press **Enter**.



5 Type `print "Hello, world\n";` and press Enter.

6 Type `print "The time is " . localtime();` and press Enter.

Note: If a `print` command does not end in a carriage-return, you can define it on the next line.

7 Save the file and exit the text editor.

Note: In Notepad, click `File → Save` to save the file; click `File → Exit` to exit the text editor.

8 Press Alt+Tab to switch to the Command Prompt.

9 In the Command Prompt, type `dir FILENAME.pl` (ls `FILENAME.pl` in Unix) to list the file.

- The directory output confirms that the file was created.

```

1 #!/usr/bin/perl
2
3
4 # Display a message to the screen
5
6 print "Hello, world\n";
7 print "The time is " . localtime();
8 print "\n";
9

```

```

H:\VB\Perl_Apache_VB\chapter6>gvim "Create a New Perl Script.pl"
H:\VB\Perl_Apache_VB\chapter6>dir "Create a New Perl Script.pl"
Volume in drive H is adamm
Volume Serial Number is 0089-0495

Directory of H:\VB\Perl_Apache_VB\chapter6

09/25/2009  09:38 AM                135
               1 File(s)                135 bytes
               0 Dir(s)  28,702,994,432 bytes free

H:\VB\Perl_Apache_VB\chapter6>

```

Extra

Perl is very lenient and forgiving of the programmer's code. Sometimes it is too forgiving and may allow bugs or problems to develop as you create more complex programs. You can use the `strict` and `warning` modules to stop the program if you create an unsafe construct, or to warn you with verbose diagnostics. You need the `use` function to load a module, and you should write it immediately below the Perl header.

Example

```

#!C:\Perl\bin\perl.exe

use strict;
use warning;

```

Optionally, you can add the switch `-w` to the Perl header in lieu of `use warning;`.

As for the choice of editing program, Perl only requires a basic text editor to manipulate its code. However, after becoming more familiar with Perl syntax, you may find that this basic text editor is too limiting, and not very efficient at producing code. More complex editors can provide additional features to make editing Perl code easier and more efficient. Features such as colored syntax highlighting, auto-indenting, and an integrated development environment (IDE) are available if you are willing to invest more time in learning to use more complex text editors.

Print Output to the Screen

Printing output to the screen is useful to convey information back to the user who is executing the program. You can use printing output to display relevant information about your program's execution, status, and results, based upon user-defined or module-defined `print` statements. You can also use printing to display debug information. You should remove or disable this content when you have completed the program.

There are a few basic variants of the `print` function that programmers commonly use to display content to the screen. The output text enclosed in double-quotes (`print "text";`) instructs Perl to expand any variables and special characters (such as `\n` for a new line) that may be included between the double quotes. Any text enclosed in single quotes (`print 'text';`) does not expand any variables or special characters; text is printed verbatim.

Print Output to the Screen

- 1 Open a Perl script in your text editor.
 - 2 Type `print` “ to begin printing text.
 - 3 Type some text, including a variable that should be expanded.
- Note:** Perl predefines the hash `%ENV` with the command line's environment.
- 4 Type `\n\n` to produce two carriage returns.
 - 5 Type “;” to end the `print` statement, and press Enter.
 - 6 Type `print` ‘ to begin a literal text string.
 - 7 Type ‘;’ to end the literal print statement, and press Enter.
 - 8 Type `print $var` to print a single variable.

Note: For a complete list of predefined Perl variables, run `perldoc perlvar`.

- 9 Type `.\n\n`; and press Enter to concatenate the variable to two carriage returns and to end the `print` statement.

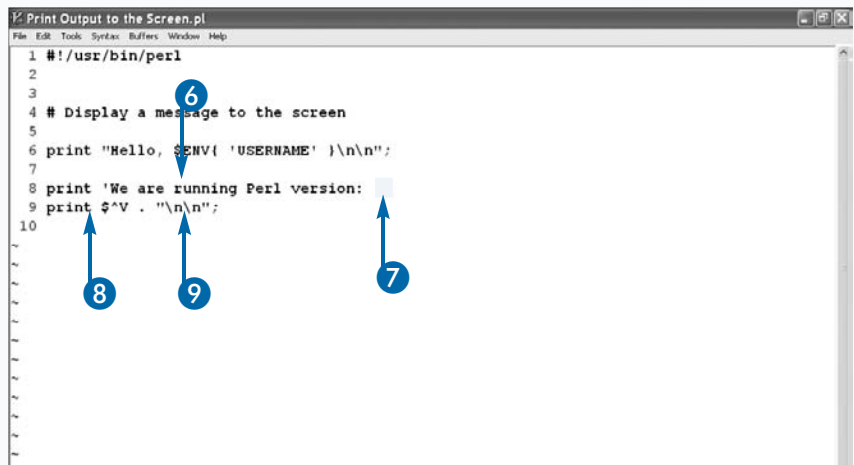
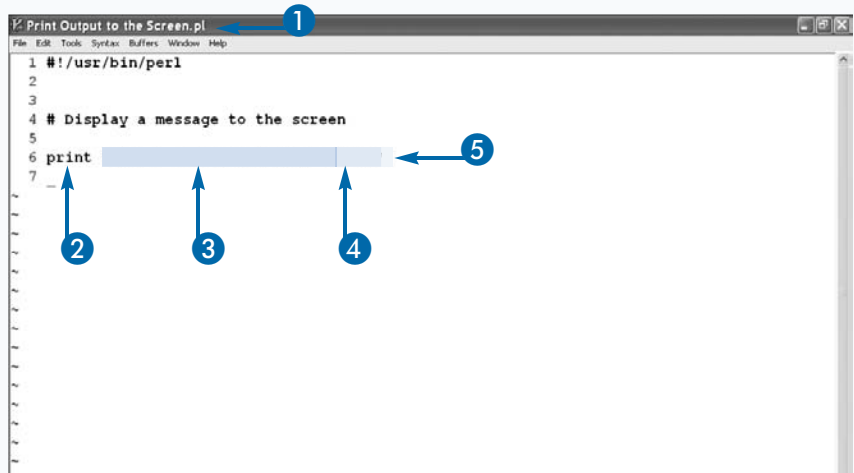
Note: The special character `\n` must be expanded to a carriage return within a double-quoted string.

Printing a number or a variable (`print value;`) does not require any quotes.

Multiple lines can be printed using a marker (`print <<EOF`), where Perl keeps reading until the marker (`EOF`) is found again on its own line. You can use any string for a marker, for example: `EOL`, `STOP`. This method also expands any variables and special characters enclosed in text. Because this allows for multiple lines of output, you no longer need the special character `\n` for a new line. Finally, you can join multiple print methods together into a single print command using the concatenation operator *period* (represented as `.`)

```
print "text" . 'text' . value;
```

When Perl sends content to CGI and HTTP to display in a Web browser, it is actually printing that information to a virtual screen.



- 10 Type **print <<EOF**; and press Enter.
- 11 Add some multi-line text and press Enter.
- 12 Type **EOF** and press Enter.

Note: Be sure to press Enter after typing EOF. If you only type the marker and save the file, you will get an error message.

- 13 Save the Perl script and exit the text editor.

- 14 Open the Command Prompt in the same directory as the Perl script.

- 15 Type **perl FILENAME.pl** on the command line and press Enter to execute the Perl script.

- The output of the Perl script appears on the screen.

```

1 #!/usr/bin/perl
2
3
4 # Display a message to the screen
5
6 print "Hello, $ENV{ 'USERNAME' }\n\n";
7
8 print 'We are running Perl version: ' ;
9 print $^V . "\n\n";
10
11 print <<EOF;
12 The runnig perl binary is: $^X
13 The running perl script is: $0
14 EOF
15

```

```

H:\VB\Perl_Apache_VB\chapter6>perl "Print Output to the Screen.pl"
Hello, adam

We are running Perl version: v5.10.1
The runnig perl binary is: C:\Perl\bin\perl.exe
The running perl script is: Print Output to the Screen.pl
H:\VB\Perl_Apache_VB\chapter6>

```

Extra

You can use the `printf` function to display formatted content to the screen, just like the `sprintf` function in C. For example, you can rewrite steps 3 and 4 as follows:

Example

```

printf "We are running
Perl version %s\n",
$^V;

```

For more information, see `perldoc -f printf`.

Perl has a special auto-flush variable called `$|`. You can use this variable to control the flow of content as it is printed out to the screen. By default, `$|` is disabled (set to 0), which means the `print` statements only actually display content when a new-line character is reached. If you enable `$|` (set it to 1), then `print` statements return data in real time.

Example

```

$| = 1;
print "This is ";
sleep( 2 );
print "the output.\n";

```

In this example, when you run your program, it literally prints “This is “, sleeps two seconds, and then prints “the output.” If you do not define `$|`, two seconds pass before the whole string displays at once. The enabling the auto-flush variable is very useful in CGI scripting because it stops Perl from waiting for a new-line character before sending anything to the browser. If you have a complex Perl script with a noticeable processing delay when constructing HTML output, you may not want to wait for the whole page to render before displaying content to the browser.

Execute a Perl Script

Every Perl script must be executed in order to run. The execution process launches the Perl interpreter component of the Perl binary, reads your script, converts it to binary data in memory, and performs the requested actions in the code. The whole process is relatively automatic, depending on your operating system.

If you are using ActiveState Perl on Windows, you do not need to follow most of the steps required to prepare a script for execution. The Perl header is largely cosmetic (but still good etiquette), and the Perl binary has already been mapped as a registered file type in Windows to the .pl extension. The Perl header is actually treated like a comment, as it still begins with a hash (#) symbol. As a result, you can execute your program by double-clicking a Perl script directly, using the command line and typing

perl *FILENAME.pl*, or even just typing the Perl script's name and running it like a stand-alone program.

If you are using Unix (or Cygwin Perl on Windows), you have two options available. First, you can specify your own Perl binary on the command line, followed by your script name. Second, you can set the execute-bit on the script using `chmod` and run the program like any binary. It is this second method that utilizes the Perl header directly. Your Unix shell opens the script and reads the first line. The Perl header is the shell's hint that this is a Perl script; it tells the shell to run the specified binary and to execute the remaining code. If you release a Perl script that you created on the Internet, it is considered good etiquette to use the default Unix path, regardless of the operating system on which you produced the code.

`#!/usr/bin/perl`

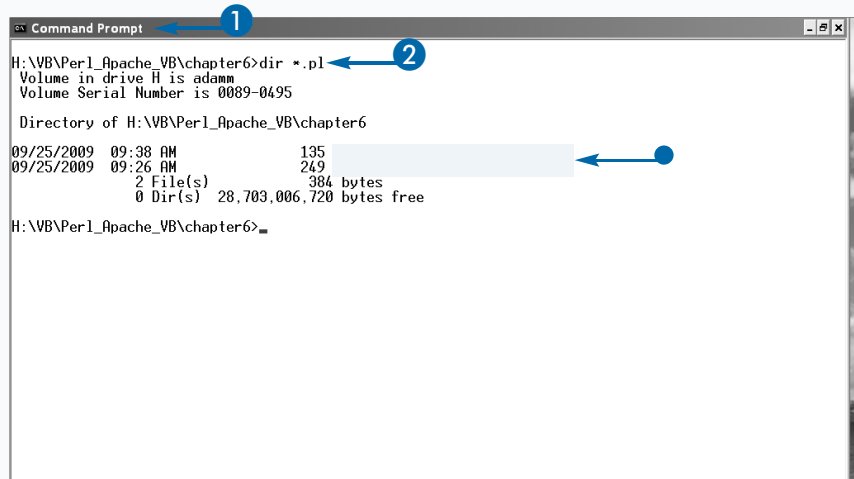
Execute a Perl Script

- 1 Open a Command Prompt to a directory with a Perl script.
- 2 Type `dir *.pl` (`ls *.pl` on Unix) to identify the script.
 - All Perl script files are in this directory.

Note: For steps on creating a Perl script, see the section, "Create a New Perl Script."

- 3 Type `C:\Perl\bin\perl.exe FILENAME.pl` (`/usr/bin/perl FILENAME.pl` on Unix) and press Enter to execute the Perl script.
 - The Perl script runs.

Note: The Perl binary is normally installed in the system's path. If so, you can run the program simply as `perl FILENAME.pl`.



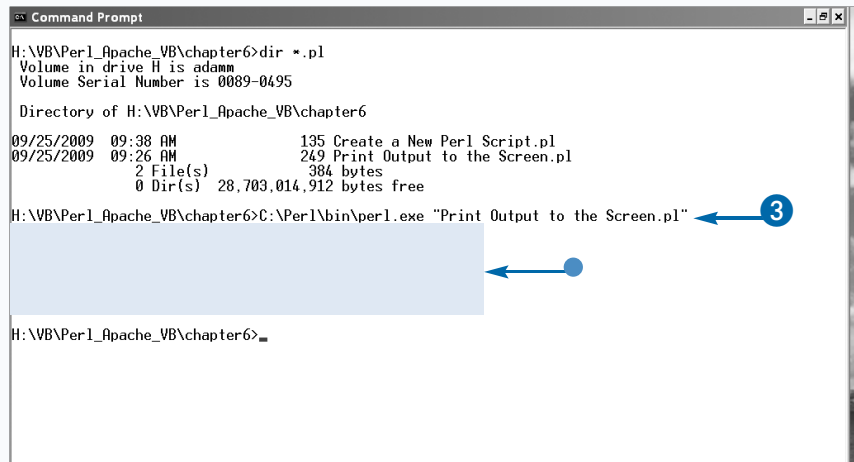
The screenshot shows a Windows Command Prompt window with the title bar "Command Prompt". The current directory is `H:\VB\Perl_Apache_VB\chapter6`. The user has entered the command `dir *.pl`. The output shows a directory listing for two files: `Create a New Perl Script.pl` (135 bytes) and `Print Output to the Screen.pl` (249 bytes). A blue circle with the number 1 points to the Command Prompt title bar, and a blue circle with the number 2 points to the `dir *.pl` command.

```
Command Prompt
H:\VB\Perl_Apache_VB\chapter6>dir *.pl
Volume in drive H is adammm
Volume Serial Number is 0089-0495

Directory of H:\VB\Perl_Apache_VB\chapter6

09/25/2009  09:38 AM                135 Create a New Perl Script.pl
09/25/2009  09:26 AM                249 Print Output to the Screen.pl
               2 File(s)                384 bytes
               0 Dir(s)  28,703,006,720 bytes free

H:\VB\Perl_Apache_VB\chapter6>
```



The screenshot shows a Windows Command Prompt window with the title bar "Command Prompt". The current directory is `H:\VB\Perl_Apache_VB\chapter6`. The user has entered the command `C:\Perl\bin\perl.exe "Print Output to the Screen.pl"`. The output shows the text "Print Output to the Screen.pl" displayed on the screen. A blue circle with the number 3 points to the command line.

```
Command Prompt
H:\VB\Perl_Apache_VB\chapter6>dir *.pl
Volume in drive H is adammm
Volume Serial Number is 0089-0495

Directory of H:\VB\Perl_Apache_VB\chapter6

09/25/2009  09:38 AM                135 Create a New Perl Script.pl
09/25/2009  09:26 AM                249 Print Output to the Screen.pl
               2 File(s)                384 bytes
               0 Dir(s)  28,703,014,912 bytes free

H:\VB\Perl_Apache_VB\chapter6>C:\Perl\bin\perl.exe "Print Output to the Screen.pl"
Print Output to the Screen.pl

H:\VB\Perl_Apache_VB\chapter6>
```

4 (Unix only) Type **chmod +x FILENAME.pl**.

Note: You only need to run the `chmod` command once before executing a script for the first time.

5 (Unix only) Type **ls -l FILENAME.pl**.

- The execute-bit is set.

6 Type **FILENAME.pl** (./**FILENAME.pl** on Unix) to execute the Perl script.

Note: If a Perl script filename contains spaces, wrap it in double-quotes. This ensures that the shell knows exactly which file you are referring to.

- The Perl script runs.

Note: An error may occur on Unix only if the execute-bit was not properly set.

```
adammm@gw: ~/VB/Perl_Apache_VB/chapter6$ ls -l *.pl
-rw-rw-r-- 1 adammm adammm 135 2009-09-25 09:38 Create a New Perl Script.pl
-rw-rw-r-- 1 adammm adammm 249 2009-09-25 09:26 Print Output to the Screen.pl
adammm@gw:~/VB/Perl_Apache_VB/chapter6$ chmod +x *.pl
adammm@gw:~/VB/Perl_Apache_VB/chapter6$ ls -l *.pl
-rwxrwxr-x 1 adammm adammm 135 2009-09-25 09:38
-rwxrwxr-x 1 adammm adammm 249 2009-09-25 09:26
adammm@gw:~/VB/Perl_Apache_VB/chapter6$
```

```
Command Prompt
H:\VB\Perl_Apache_VB\chapter6>dir *.pl
Volume in drive H is adammm
Volume Serial Number is 0089-0495

Directory of H:\VB\Perl_Apache_VB\chapter6

09/25/2009  09:38 AM                135 Create a New Perl Script.pl
09/25/2009  09:26 AM                249 Print Output to the Screen.pl
               2 File(s)              384 bytes
               0 Dir(s)  28,703,014,912 bytes free

H:\VB\Perl_Apache_VB\chapter6>C:\Perl\bin\perl.exe "Print Output to the Screen.pl"
Hello, adammm

We are running Perl version: v5.10.1

The running perl binary is: C:\Perl\bin\perl.exe
The running perl script is: Print Output to the Screen.pl
H:\VB\Perl_Apache_VB\chapter6>"Print Output to the Screen.pl"
```

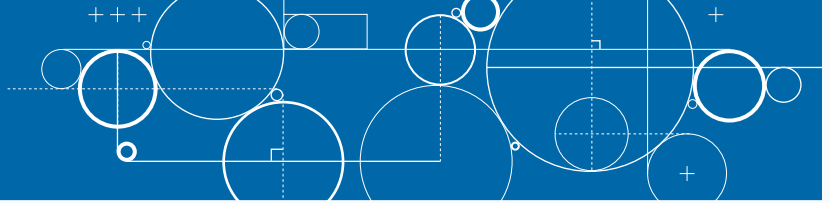
Extra

It is possible to actually embed the Perl binary and your script directly into a single file. You can do this using the `perlcc` program. If it is installed, simply run `perlcc scriptname` to produce `scriptname.exe`. You can then place this stand-alone executable onto any computer that supports the output binary type and run it, even if Perl is not installed.

While `perlcc` does allow for the convenience of deploying custom-compiled code without deploying Perl (or the original script) on foreign systems, do not think of this as being any more secure. If your original script contained any sensitive data (such as hard-coded passwords), a skilled user can still retrieve that data, even if the average person were to open the compiled binary into Notepad and see nothing but binary gibberish.

Also remember that the output produced by `perlcc` actually has the entire Perl interpreter embedded directly within the binary. This could make a simple program that only prints one or two lines that appear to be larger than 10MB in size.

Introducing Perl Scalars



Scalars are the most primitive variable type available to Perl, representing a single value. A scalar may contain a character, a string, an integer, or a floating number. Scalars do not require complex declaration rules to dictate their size or type, as in C or C++, but it is a good practice to declare them with the `my` command to set their scope.

A scalar's value is free to change from undefined to text, to integer, and back to undefined. If you need to know what type of data a scalar holds, the Perl CPAN module `Data::Types` provides a means of identifying and converting a scalar's type. Chapter 9 contains more information about installing CPAN modules.

A scalar can also store a reference to another variable type, such as another scalar, array, or hash — or, in other words, a `scalarref`, `arrayref`, or `hashref`. This is roughly the Perl equivalent to pointers in C or C++.

By nesting variable references together, you can construct very complex data structures anchored by a single scalar variable. That single scalar can be passed to functions, modules, files — almost anywhere — and the entire complex data structure comes along with it.

Modules actually use scalar references as a doorway into their library of functions and methods. Depending on the module, you may need to initialize the module into a scalar variable, which can be used to reference any of the code in that particular instance of the module.

Declaring Scalars

You declare a scalar using the `my` command, regardless of the type of data it will hold. The location of the declaration restricts the scope of the scalar, based upon the enclosing block. It is a good practice to declare all variables at the start of the block.

```
my $scalar;
```

When declaring a scalar, or any type of variable, it is possible to assign a value to it on the same line.

```
my $scalar = value;
```

Additionally, `my` supports declaring and assigning multiple variables all on one line by wrapping the declaration and assignment portions with parenthesis.

```
my ( $scalar1, $scalar2, ..., $scalarN ) = ( value1, value2, ..., valueN );
```

Keep in mind that when using parenthesis with `my`, the number of items on both sides of the equal sign should be the same. If there are more variables declared than values assigned, additional variables will be declared but remain undefined. If there are more values than variables, then additional values will be silently ignored.

Remember, re-declaring a variable within a sub-block does not affect its previous value, but re-assigning a previously declared variable within a sub-block does.

Storing Data in Scalars

Scalars can hold any type of data, and you can assign them with an equal sign. Data on the right side of the equal sign is copied and saved into the variable on the left side.

```
$scalar = value;
```

In this example, the data is copied from `value` into `$scalar`. If the source `value` is another scalar and its value changes, then the original `$scalar` is unaffected. If you want to have access to the same value, then you need a reference.

Retrieving Data in Scalars

Accessing data in scalars is very simple. A scalar can store anything you ask it to: a text string, an integer, a floating number, or even a reference to more complex data such an `arrayref`, a `hashref`, or even a module.

Printing scalars, comparing scalars, or simply passing their value to other functions implies retrieving their data.

Retrieving Referenced Data in Scalars

Accessing referenced data in scalars is a little different from accessing normal data. Using the previous example, if you simply call `print $ref;`, it displays something like this:

```
SCALAR(0x8f2f0d8)
```

Remember, you stored into `$ref` a pointer to the data, not the actual data. You literally see here the location of your data by its memory address.

To properly *dereference* a variable and access its original value, you need a second `$`. There are two ways to write the code for this:

```
print $$ref;
print ${ $ref };
```

While both methods produce the same results, the second method is technically more correct, and a better habit to get into. This is simply an example of a one-dimensional reference. When dealing with nested references of three or four dimensions, the second long-handed dereference technique makes it easier to access deeply buried data.

Storing Referenced Data in Scalars

The difference between assigning an actual value or a referenced value to a scalar is that assigning an actual value makes a copy of the value directly, while assigning a referenced value makes a copy of the memory location of the value. The memory location is useful because it acts as a pointer to the original data. After a variable has been declared, its memory location does not change, even though its value might.

You can create references by preceding the target with the backslash (`\`) character when assigning its value to a scalar.

```
my $scalar = value;
my $ref = \ $scalar;
```

If `$scalar` were to change, `$ref` (when retrieved correctly) would provide access to `$scalar`'s new value, provided that `$ref` and `$scalar` remain correctly scoped.

Destroying Scalars

Perl does not expect the developer to explicitly free scalars from the system memory, or any other type of variable. When the block of code ends that originally declared the variable, the memory is automatically freed.

However, sometimes you might want to proactively undefine a value at a particular point in the code; in this case, you can use `$scalar = undef`. This keeps the scalar active in memory, but removes its contents.

To proactively forget that a scalar ever existed, you can use the `undef($scalar)` command. Remember, this happens anyway once the block that originally declared the scalar ends.

Store Data into Scalars

Scalars are the most primitive variable type in Perl, representing a single value. Scalars store any data type, including an integer, floating number, text, or even references to more complicated variables.

All scalars begin with a dollar sign (\$), followed by one or more alphanumeric characters. (User-defined scalars must never begin with a number.)

You declare all variables with the `my` command. You can pick a variable name and assign a value to it without declaring it, but this is not a good coding practice. Note that the `strict` module enforces declared variables with `my`.

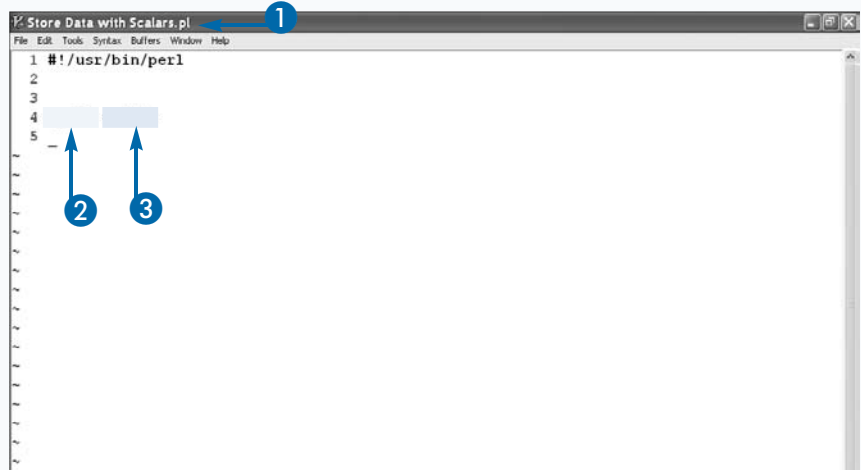
You do not need to establish a scalar's type when declaring. It can dynamically switch between undefined, number, text, and reference, just by re-assigning its value.

Store Data into Scalars

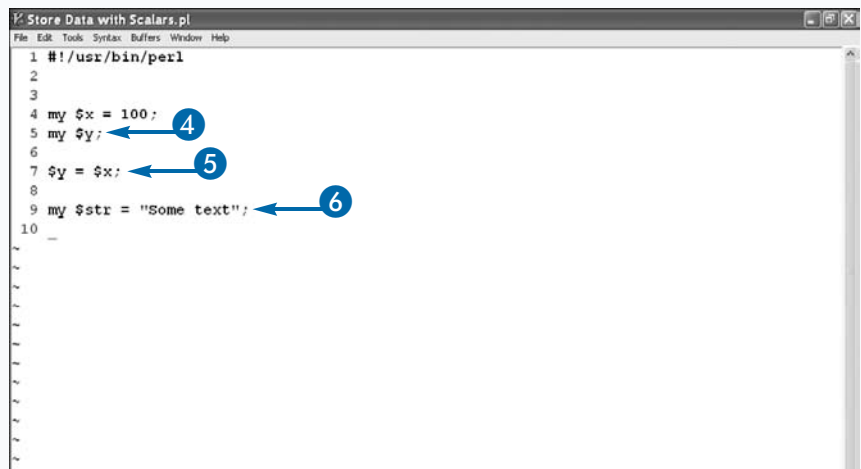
- 1 Open a Perl script in your text editor.
- 2 Type **`my $scalar`** to declare a new scalar variable.
- 3 Type **`= VALUE`**; to assign a value to the scalar on the same line and end the statement.
- 4 Type **`my $scalar`**; to declare a new scalar variable and end the statement.
- 5 Type **`$scalar = VALUE`**; to assign a value to an existing variable.
- 6 Type **`my $scalar = "TEXT"`**; to declare and assign a scalar with text in one statement.
- 7 Save the Perl script.

Extra

When storing data into scalars, pay attention to the block of code that housed the scalar's declaration. Regardless of where you assign its value, data is only available in the original block that established the variable's declaration. Activating the `strict` module in a script will stop you if you attempt to assign a value to an undeclared scalar. By not using `strict`, you can avoid this error. Activating the `warning` module in a script warns you if you accidentally re-declare an existing variable. Re-declaring a scalar in the same block of code with `my` only has the effect of re-assigning its value. Re-assigning a scalar does not require `my`, as long as the update exists at or below the block that housed the declaration.



```
1 #!/usr/bin/perl
2
3
4 my $x;
5 = 100;
```



```
1 #!/usr/bin/perl
2
3
4 my $x = 100;
5 my $y;
6
7 $y = $x;
8
9 my $str = "Some text";
10
```


Retrieve Data from Scalars

You can access data from a scalar by simply citing the scalar in the context of where it is needed. Just like when storing data, you always prefix scalar variables with the dollar sign (\$) when reading data.

A scalar's value may be defined or undefined. A scalar is deemed *defined* if a value has been assigned to it after it has been declared. A scalar is *undefined* if no value has been assigned, or has never been properly declared.

When retrieving data from a scalar, pay attention to the block of code that housed the scalar's declaration. Regardless of where you access its value, the data is only available in the original block that established the variable's declaration.

Apply It

Using the concatenation operator (a single period) instructs Perl to print the two parts as a single string:

```
print $x . ", " . $y . ": " . $str . "\n";
```

Alternatively, you can avoid the use of the concatenation character by accessing all variables in a single string:

```
print "$x, $y: $str\n";
```

Because variables cannot contain a comma or a colon, Perl interprets `$x` and `$y` correctly.

Retrieve Data from Scalars

- 1 Open a Perl script in your text editor.
- 2 Type `print "$scalar\n";` to display a scalar's value and a new line.
- 3 Type `print $scalar . "$scalar\n";` to display the value of two scalars, concatenated together, and a new line.
- 4 Save the Perl script.

```

1  #!/usr/bin/perl
2
3
4  my $x = 100;
5  my $y;
6
7  $y = $x;
8
9  my $str = "Some text";
10
11 print "$x\n";
12 print $y . " $str\n";
13 print "\n";
14
~
~
~
~
~
~
~
~
~
~

```

- 5 Open a Command Prompt in the same directory as the Perl script.
- 6 Execute the Perl script.
 - The output of the Perl script appears.

```

H:\VB\Perl_Apache_VB\chapter6>perl "Store Data with Scalars.pl"

```

Introducing Perl Arrays

An array variable in Perl is basically a list of values that are saved in a specific order. Each value in the array is can be accessed by its *index* number, reflecting the order in which new data was added. An actual array is not usually complicated; it inherits its complexity from each value that is defined within.

The simplest form of an array is one that only stores basic scalar values, and keeps track of their order. A more complex array might store other arrays, hashes,

scalarrefs, arrayrefs, or hashrefs as a value. By layering arrays on top of each other, you can create a multi-dimensional array.

The actual array structure is very malleable. Functions exist that allow it to dynamically grow and shrink based upon the developer's requirements. You can remove groups of one or more elements from the middle and place them at the end. Individual values of an array are manipulated just as easily. It is a very flexible way of storing information in a linear format.

Declaring Arrays

You declare an array using the `my` command, regardless of the type of data or the number of elements it will hold. The location of the declaration restricts the scope of the array, based upon the enclosing block. It is a good practice to declare all variables at the start of a block.

```
my @array;
```

When declaring an array, or any type of variable, it is possible to assign a value to it on the same line. Remember, re-declaring a variable within a sub-block does not affect its previous value, but re-assigning a previously declared variable within a sub-block does.

Declaring a Referenced Array

A *referenced array* (`arrayref`) differs from a normal array in that it provides only a pointer to the array data. As a result, you need a scalar to hold the pointer.

```
my $arrayRef = [];
```

The primary benefit of using referenced arrays over normal arrays occurs when passing data through functions. Because functions accept only one anonymous array as input parameters, passing one `arrayref` acts as a single scalar from the function's perspective. If you have multiple data lists to send to a function, you must use references. If you use regular arrays, the function will not be able to identify when the first array ends and the second one begins!

Storing Data into Arrays

You can store data into arrays in two ways: manually or with a helper function.

Manually Setting Array Data

If you are declaring an array, and you know its initial values, then you can do this on one line.

```
@array = ( value1, value2, ..., valueN );
```

If you have already declared the array, and you are privy to the total size of the array, then you can manually specify the element to update by its index.

```
$array[ index ] = value;
```

If you do not know the current length of an array, or you want to purposely add a value to an arbitrary location, then you can do that. Any earlier values that were not used are now declared, but undefined.

When using the manual method, if data already existed at a particular index, then it is over-written without warning.

Storing Data into Arrays *(continued)*

Using a Function to Set Array Data

Perl provides various functions to inject data into an array at the beginning, middle, or end of the list. Using a function avoids accidentally losing existing data, especially at the beginning or middle. You can use the `push` function to append data to the end of an array.

```
push( @array, value );
```

Use `unshift` to insert data at the start of an array.

```
unshift( @array, value );
```

A way to simply parse the contents of a string into an array is to use the `split` command.

```
my @array = split( delimiter, string );
```

Retrieving Data from Arrays

As with storing, you can retrieve data from arrays in two ways: manually or with a helper function.

Manually Accessing Array Data

Manually accessing data in an array is exactly like setting it; you only need to know the element index.

```
print $array[ index ];
```

Arrays are zero-indexed, meaning that the first value is at position 0, the second value is at position 1, and so on. You can identify the last element of an array (or the array's length) using the special variable `$#array`. You can access the total number of elements in an array using the original array variable's name but in a scalar context.

```
if ( @array > $count ) { ... }
```

Manually Using a Function to Access Array Data

The easiest way to process all data in an array is with the `foreach` loop. Each element in the array gets one pass of

the following code block. The special variable `$_` is conveniently set to the currently active value.

```
foreach ( @array ) {
    print $_;
}
```

A minor variance of a `foreach` loop counts by index value, rather than the actual value.

```
foreach ( 0..$#array ) {
    print $array[ $_ ];
}
```

Both examples print the exact same thing, the difference being that the first sets `$_` to each array element value, and the second sets `$_` to each array element index. A way to simply merge the contents of an array into a string is to use the `join` command.

```
print join( ',', @array );
```

Manipulating Data in Arrays

You can manipulate data that is stored in arrays using the functions `pop`, `shift`, and `splice`. In fact, `pop` and `shift` are the exact opposites from `push` and `unshift`, which were described earlier. The `pop` function removes the last element from the end of an array, while the `shift` removes it from the start of the array. Both of these functions return as output the value they removed; the array shrinks as necessary. The `splice` function allows you to add or remove one or more elements in an array, at any position in the array.

Destroying Arrays

You can remove individual elements of an array using `delete($array[$index])`. However, deleting an element does not reorganize the array by shuffling everything around the empty index. Instead, the element at that index becomes undefined.

When completely emptying an array, `delete` is rather inefficient. Instead, you should use `@array = ()`. To proactively forget that an array ever existed, you can use the `undef(@array)` command. Remember, this happens anyway once the block that originally declared the array ends.

Store Data into Arrays

Arrays are a series of zero-indexed values that are available to Perl. Think of them as a set of scalars, each assigned a number representing the order in which they are added. You can append data into an array using the `push` command. This can be useful if you do not know exactly how big the array is, or if you have large amounts of data to append.

When accessing individual elements in an array, you must prefix the array variable name with a dollar sign (\$), just like a scalar. When acting upon the array as a whole (declaring, re-assigning, and so on), you must prefix the variable name with an ampersand (&).

Store Data Into Arrays

Extra

Perl includes some functions for storing data in arrays. You can use the functions `push` and `unshift` to insert one or more values into the end and start of an array, respectively. You can use the function `splice` to add, replace, or remove a series of elements in an array.

These functions are helpful because manually setting a value by its index is only recommended if you already know exactly how many elements exist in the array.

- 1 Open a Perl script in a text editor.
- 2 Type **my @array** to declare a new array.
- 3 Type **= (** to begin assigning values to @array.
- 4 Assign comma-separated values to the array.
- 5 Type **);** to close the array assignment statement, and then press Enter.

```
1 #!/usr/bin/perl
2
3
4 my @animals = ( 'cat', 'dog', 'bird' );
5
```

Diagram annotations: 1 points to the file name 'Store Data into Arrays.pl'. 2 points to 'my'. 3 points to '@animals'. 4 points to the first comma in the list. 5 points to the closing parenthesis and semicolon.

- 6 Type **push(@array,** to append a new value into the array.
- 7 Add a new value into the array.
- 8 Type **);** to close off `push()`'s arguments and the statement, and then press Enter.
- 9 Type **\$array[2] =** to re-assign the value in the second index.
- 10 Assign a new value to replace the second index's value, and then press Enter.
- 11 Save the Perl script.

```
1 #!/usr/bin/perl
2
3
4 my @animals = ( 'cat', 'dog', 'bird' );
5
6 push( @animals, 'rabbit' );
7
8 $animals[ 2 ] = 'mouse';
9
```

Diagram annotations: 6 points to 'push('. 7 points to the opening parenthesis of the push function. 8 points to the closing parenthesis and semicolon of the push statement. 9 points to '\$animals'. 10 points to the value 'mouse'.

Retrieve Data from Arrays

When retrieving individual elements out of an array, you must prefix the array variable name with a dollar sign (\$), just like a scalar. When acting upon the array as a whole (updating, listing, and so on), you must prefix the variable name with an ampersand (&).

You can easily access the entire contents of an array using the `foreach` command. If you are interested in an individual value, you can reference the array directly by the value's index position. The `foreach` loop is a useful tool for retrieving data from an array, especially if you do not know exactly how many elements are in the array. To access the total length of an array, you can prefix the variable name with a dollar sign and a hash symbol (\$#).

Apply It

Normally, when `$_` is populated within a `foreach` loop, you only use it for reading the current array element. However, you may assign a new value into it, which updates the original array at the current array element. It is possible to loop on by the array's index directly.

```
foreach ( 0..$#pets ) {
    print "Pet " . $_ . ": " . $pets[ $_ ]
    . "\n";
}
```

This code instructs `foreach` to loop from 0 to the number of pets, setting the current index as `$_`.

Retrieve Data from Arrays

- 1 Open a Perl script in a text editor.
- 2 Type `foreach (@array) {` to create a new `foreach` loop block over the array, and then press Enter.
- 3 Print the element with `$_`.
- 4 Type `}` to close the `foreach` loop block, and press Enter.
- 5 Type `$array[0]` to access the first indexed element from the array.
- 6 Save the Perl script.
- 7 Open a Command Prompt in the same directory as the Perl script.
- 8 Execute the Perl script.
 - The output of the Perl script appears.

```
1 #!/usr/bin/perl
2
3
4 my @animals = ( 'cat', 'dog', 'bird' );
5
6 push( @animals, 'rabbit' );
7
8 $animals[ 2 ] = 'mouse';
9
10 foreach ( @animals ) {
11     print "Found animal: " . $_ . "\n";
12 }
13
14 print "The first animal in the array is a " . $animals[ 0 ] . "\n";
15
```

```
H:\VB\Perl_apache_VB\chapter6>perl "Store Data into Arrays.pl"

Found animal: cat
Found animal: dog
Found animal: bird
Found animal: rabbit
Found animal: mouse
The first animal in the array is a 1

H:\VB\Perl_apache_VB\chapter6>
```

Introducing Perl Hashes

A hash variable in Perl is almost exactly like an array, except that rather than storing values in a series with a number, it uses an arbitrary key. (Hashes are often referred to as a set of key/value pairs.)

Like an array, the simplest form of a hash is one that stores only scalar keys and scalar values. A more complex hash might store other arrays, hashes, `scalarrefs`, `arrayrefs`, or `hashrefs` as a value, or even a key!

For example, a simple hash can be represented as a store's inventory of car parts. Each part has a name, which is stored as the key in the hash. The available count of each part is stored as each key's respective

value. A more complex hash of car parts may still use the part's name as a key, but the value could be another hash representing the inventory, retail price, wholesale price, contact information, orders, and so on.

Like arrays, the hash structure is very flexible and easily influenced. Perl dynamically manages its memory usage based upon its size. Hashes are easier to manually manipulate than arrays, and as a result, they require fewer built-in functions to control.

Hashes are a very powerful and efficient method of storing and retrieving information in a non-linear format. It is important to not confuse hashes with hash marks (`#`), the latter being used only for comments in your code.

Declaring Hashes

You declare a hash using the `my` command, regardless of the type of data or the number of keys it will hold. The location of the declaration restricts the scope of the hash, based upon the enclosing block. It is a good practice to declare all variables at the start of a block.

```
my %hash;
```

When declaring a hash, or any type of variable, it is possible to assign a value to it on the same line. Remember, re-declaring a variable within a sub-block does not affect its previous value, but re-assigning a previously declared variable within a sub-block does.

Declaring a Referenced Hash

A *referenced hash* (`hashref`) differs from a normal hash in that it provides only a pointer to the hash data. As a result, you need a scalar to hold the pointer. You can rewrite the previous example as:

```
my $hashRef = {};
```

Like arrays and `arrayrefs`, the primary benefit of using referenced hashes over normal hashes arises when passing data through functions. Because functions accept only one anonymous array as input parameters, passing one `hashref` acts as a single scalar from the function's perspective. If you have multiple data lists to send to a function, then you must use references. If you use regular hashes, then the function cannot identify when the first hash ends and the second one begins!

Storing Data into Hashes

Storing data into a hash takes two forms. The first form allows the developer to replace an entire hash, and set multiple key/value pairs in one command. The second form allows the developer to append a new key/pair onto an existing hash, without affecting earlier assignments.

Multiple Key/Value Pair Assignments

You should only assign multiple key/value pairs in a single statement when initially declaring a new hash variable. If you use this method on an already-defined hash variable, then all content will be replaced by the new key/value pair assignment group.

```
%hash = (  
    key1 => value1,  
    key2 => value2,  
    ...  
    keyN => valueN,  
);
```


Storing Data into Hashes *(continued)*

Note that the last key/value pair may optionally end in a comma. Perl allows this to make it easier for the programmer to add content to the end of the last line (for example, by copying and pasting the last line multiple times). Also, if the last assignment is commented out, but the previous assignment comma remains, then it should not trigger an unnecessary error message.

Single Key = Value Assignment

Using a simple “key = value” statement allows you to manipulate a single key in a hash without affecting any other keys. If a particular key already exists, then its value will be updated.

```
$hash{ key } = value;
```

Only one key can be updated per statement using this method.

Retrieving Data from Hashes

If you know a particular key in the hash, then it is relatively easy to access data from it. If you do not know what keys exist, then you need some sort of discovery process.

Reading a Value from a Known Key

Reading an exact value is just like assigning an exact value. Only a single key/value pair can be read using this method.

```
print $hash{ key };
```

If *key* does not exist, then the data returned will be undefined.

Retrieving a List of Keys

If you do not know the keys, then you can retrieve a list of all available keys into an array.

```
my @allKeys = keys %hash;
```

From here, @allKeys can be processed like any other array through another function such as `foreach`.

Retrieving a List of Values

If you are only interested in the contents of the hash, then it is possible to bypass the keys step entirely.

```
my @allValues = values %hash;
```

Again, you can use `foreach` to display all values in the array.

Reviewing Complex Hashes

You can use the `Data::Dumper` module to review the contents of a hash if it becomes too complex, or if a hash contains data that you were not expecting to see.

```
use Data::Dumper;
CODE;
print Dumper( VAR );
```

In fact, you can use `Dumper` to reference any type of variable, not just a hash. This will be especially useful later when dealing with nested hash references, as described in Chapter 8.

The `Perldoc` program provides details on this powerful module. Type the command **`perldoc Data::Dumper`** for more information.

Destroying Hashes

You can remove individual elements of a hash using `delete($hash{ $key })`. When completely emptying a hash, `delete` is rather inefficient. Instead, you should use `%hash = ()`.

To proactively forget that a hash ever existed, you can use the `undef(%hash)` command. Remember, this happens anyway once the block that originally declared the array ends.

Store Data into Hashes

Hashes are similar to arrays, except that where arrays use a numbered index to reference the value, hashes use a key. Unlike arrays, there are no additional functions required to build a hash.

When initially declaring a hash, you can assign multiple key/value pairs in a single statement. After a hash has been declared, you can only assign a single key/value within a single statement. If you try to assign multiple values to an already-declared array, then its previous content will be lost.

When accessing individual elements in a hash, you must prefix the hash variable name with a dollar sign (\$), just like a scalar. When acting upon the hash as a whole, you must prefix the variable name with a percent sign (%).

Store Data into Hashes

- 1 Open a Perl script in a text editor.
- 2 Type **my %hash = (** to declare a new hash.
- 3 Type **key => value**, for each key/value pair in the hash, and press Enter after each pair.
- 4 Type **);** to close the hash assignment, and press Enter.

***Note:** It is acceptable for the last key/value pair in a hash assignment to end with a comma, even though there is nothing following it.*

- 5 Type **\$hash** to begin to assign an individual value.
- 6 Type the key that will be affected.
- 7 Type **= value** and assign its new value.
- 8 Type **;** to close the assignment statement, and press Enter.
- 9 Save the Perl script.

Extra

You can assign any type of data into a hash, including a scalar, an array, or even another hash. Once you start nesting variables together, it is possible to lose track of how your hash is built, if you forget its structure.

A useful tool for reviewing the contents of a hash (or any other type of variable) is the `Data::Dumper` module. You can use its `Dumper` function to display its contents, no matter how complex.

The `Perldoc` program provides details on this module. Type the command `perldoc Data::Dumper` for more information.

```
1 #!/usr/bin/perl
2
3
4 my %books = (
5
6
7
8 );
9
```

Diagram illustrating the initial hash declaration. Numbered callouts point to the following parts of the code: 1 points to the file name 'Store Data into Hashes.pl'; 2 points to the opening parenthesis of the hash assignment; 3 points to the opening curly brace of the hash; 4 points to the closing parenthesis and semicolon of the hash assignment.

```
1 #!/usr/bin/perl
2
3
4 my %books = (
5     'computer manuals' => 7,
6     'fiction' => 21,
7     'non-fiction' => 10,
8 );
9
10 $books{'autobiography'} = 2;
11
```

Diagram illustrating the hash being populated with values. Numbered callouts point to the following parts of the code: 5 points to the opening curly brace of the hash; 6 points to the key 'computer manuals'; 7 points to the value 7; 8 points to the closing parenthesis and semicolon of the hash assignment; 9 points to the opening curly brace of the hash; 10 points to the key 'autobiography'; 11 points to the value 2.

Retrieve Data from Hashes

You can easily access the entire contents of a hash through the `while` and `each` commands, or through the `foreach` command. If you are interested in an individual value, then you can access the hash directly by referencing the value's key.

When retrieving individual elements out of a hash, you must prefix the hash variable name with a dollar sign (\$), just like a scalar. When acting upon the hash as a whole (updating, listing, and so on), you must prefix the variable name with a percent sign (%).

To access all keys in a hash, you use the `keys` command. To access all values (ignoring the key entirely and treating the hash like an array), you can use the `values` command. For more information on looping, see Chapter 7.

Apply It

Data in a hash can be transliterated into different formats. You can change the `while` loop to a `foreach` loop:

```
while ( keys %books ) {
    print $_ . ": " . $books{ $_ } . "\n";
}
```

You can create a book counter using the `values` function:

```
my $count = 0;
foreach ( values %books ) {
    $count += $_;
}
```

Retrieve Data from Hashes

- 1 Open a Perl script in a text editor.
- 2 Type `while (my ($key, $value) = each %hash) {` to provide structure to the loop.
- 3 Print the key and value, using the scalars `$key` and `$value`.
- 4 Type `}` to close the `while` loop block.
- 5 Use `$hash{key}` to retrieve an individual element from `%hash`.

Note: Remember that an individual element in a hash is represented as a scalar; use a dollar sign.

- 6 Save the Perl script.
- 7 Open a Command Prompt in the same directory as the Perl script.
- 8 Execute the Perl script.
 - The output of the Perl script appears.

```
1 #!/usr/bin/perl
2
3
4 my %books = (
5     'computer manuals' => 7,
6     'fiction' => 21,
7     'non-fiction' => 10,
8 );
9
10 $books{ 'autobiography' } = 2;
11
12 while ( my ( $key, $value ) = each %books ) {
13     print "book count by type: $key = $value\n";
14 }
15
16 print "\n";
17 print "There are $books{ 'fiction' } books in the fiction hash key\n";
18
19 ~
20 ~
21 ~
```

```
H:\VB\Perl_Apache_VB\chapter6>perl "Store Data into Hashes.pl"
book count by type: computer manuals = 7
book count by type: fiction = 21
book count by type: non-fiction = 10
There are 21 books in the fiction hash key

H:\VB\Perl_Apache_VB\chapter6>
```

Introducing Perl Conditions

You use Perl conditions as tests to validate data as it progresses through your Perl script. You will most often find Perl conditions within parenthesis immediately following *statement modifiers*: statements that begin with commands like `if`, `elsif`, and `unless` which introduce a sub-block of code to be executed only if the condition is true.

Conditions are often split into three parts: a value, an *operator* — a simple command that mathematically or logically compares the two values — and another value. A value may be a variable, a function, a subroutine, or a hard-coded integer or string. For example, the conditional test, `if ($x > 5)` is true if `$x`'s value is greater than five.

Sometimes, conditions may only have a single part: a value. In this case, Perl is assuming that you want to determine that this value is defined and not equal to zero. For example, `if ($x)` is true if `$x` has been declared and contains something other than the number zero.

If a condition is true, then a specific block of code is executed. If the condition is false, then another block of code is executed, or nothing at all. This method of splitting the flow of the code based upon conditional results is a fundamental component of all modern programming languages.

About Statement Modifiers

You use a statement modifier to alter the flow of code based upon the results of a conditional test expression. The most common statement modifier is `if (EXPR)`, which is `if` followed by a conditional test expression in parenthesis, followed by a code block. In other words, if the conditional test expression is true, then Perl should follow the code block when running your program.

It is possible to create a *compound statement* — multiple statement modifiers and conditional tests — by appending an `else` or `elsif (EXPR)` statement modifier after the original `if (EXPR)` code block. If the original

conditional test expression fails, then code after `else` is executed. An `elsif` acts just like `else` except that it provides for a way to add another conditional test expression.

It is possible to specify multiple statement modifiers within a compound statement; it must begin with `if (EXPR)`, it may have multiple `elsif (EXPR)` code blocks, and it may end in exactly one `else`. In other words, `elsif` and `else` are optional, but an `elsif` can never follow an `else` in the same compound statement. A less common statement modifier is `unless`. In essence, `unless (EXPR)` is exactly like `if (! EXPR)`.

Syntax of Conditional Tests

Perl enables three different ways to issue a conditional test and subsequently execute code. The following three examples do the exact same thing, but are just written differently.

Test Condition, Execute Code

The most common conditional test syntax starts with the `if` statement modifier, followed by one or more tests in brackets as the expression, and then the code block to be executed within curly brackets.

```
if ( $x >= 0 ) {  
    print "x is a positive number\n";  
}
```

Define Code, Test Condition, Execute

It is possible to reverse the syntax of the first example to define the code first, and then provide the `if` statement modifier and conditional test expression. This method uses fewer lines of code, but does make the syntax slightly more convoluted. If the conditional test is true, then the portion of the line before the test is executed.

```
print "x is a positive number\n" if ( $x >= 0 );
```

Syntax of Conditional Tests *(continued)*

Execute Code, Test Condition, Execute Code

The final method does not use `if` at all, but the *logical-and* (`&&`) or *logical-or* (`||`) operators. When a *logical operator* is found between two statements, Perl executes the first

one as-is, and then, based upon the operation used and the results, optionally executes the second statement.

```
( $x >= 0 ) && print "x is a positive
number\n";
```

Nesting Conditions

Nesting conditions allow the developer to test more than one condition using *logical-and* (`&&`) and *logical-or* (`||`) operators, all within a single `if` or `elsif` statement modifier. The syntax is exactly the same as the “Execute Code, Test Condition, Execute Code” example given earlier, except it is now contained within the conditional test expression itself.

```
if ( ( $x >= 0 ) && ( $y >= 0 ) && ( $z >= 0 )
) {
    print "x, y, and z are all positive
numbers\n";
}
```

Note the additional set of parenthesis around the new logical-and tests. This is a requirement by the parent `if` command and establishes an order for Perl to digest the

expressions and execute the conditional test correctly. Because all portions are true, it must execute the code block.

To add on another layer of nested conditions within this example, introduce another set of brackets around the tests that should take precedence.

```
if ( ( ( $x >= 0 ) && ( $y >= 0 ) && ( $z >= 0
) ) ||
    ( ( $x < 0 ) && ( $y < 0 ) && ( $z < 0
) ) ) {
    print "Either x, y, and z are all
positive numbers, or\n";
    print "x, y, and z are all negative
numbers\n";
}
```

Looping Conditions

Looping conditions allow you to execute code repeatedly while a test condition is true. You can do this with either the `while (EXPR)` or `until (EXPR)` statement modifiers. Using this type of testing loop allows for additional work in the following code block to persistently adjust the original conditional test until it becomes false.

```
while ( $x < 0 ) {
    print "x is negative, it must be a
positive number,\n";
    print "now incrementing x's value\n";
    $x++;
}
```

If a looping condition never adjusts the original test condition, or never causes the original test condition to return false, then an *infinite loop* occurs. This results in code that never exits a code block and actually appears to the user to have frozen or crashed. It is important to avoid looping conditions unless you are absolutely sure the process flow can continue. Nesting looping conditions is acceptable, just like in the `if` or `elsif` statement modifiers.

Introducing Perl Operators

You use Perl operators to compare, manipulate, control, and execute code on variables, values, subroutines, and other operators. There are multiple types of operators available in Perl, such as *mathematic operators* (for adding, subtracting, and multiplying), *assignment operators* (for setting variables), and *relational operators* (for comparing test conditions).

About Operator Precedence

Operator precedence is the built-in way for Perl to evaluate some operators before others. You can alter Perl precedence by wrapping a conditional test in brackets, forcing it to run first. You can find the complete list of Perl operators and precedence using the PerlDoc command, `perldoc perlop`.

Assignment Operators

You use *assignment operators* to store values into variables in Perl. The simplest assignment operator is the equal sign (=) where the value on the right side is stored in the variable on left side. Perl supports more complex assignment operators where a variable may be read and updated in one statement.

```
$x = 2;           # Store the number 2 in $x
$y += 5;          # Add 5 to $y
$z **= 3;         # Multiply $z to the power of 3
( $z * $z * $z )
$message .= " more text";      # Append text
to $message
```

You can combine any mathematic, string, or bitwise operator with an assignment operator. Note that `~=` may appear to be an assignment operator, but it is actually a regular-expression operator. See the section, “Regular-Expression Operators,” for more information.

Mathematic Operators

You use mathematic operators to perform simple or complex calculations on numbers. Along with standard math characters, Perl has some of its own definitions for other operations. More complex functions are provided by the `Math::Complex` module.

OPERATOR(S)	DESCRIPTION	EXAMPLE
+, -, *, /	Add, subtract, multiply, divide operators	<code>\$x = (\$a + \$b) * \$c;</code>
**	Exponentiation operator	<code>\$x = \$num ** \$power;</code>
%	Modulus operator	<code>\$x = \$c % 5;</code>
++, --	Increment or decrement by one operators	<code>\$x++; \$x--;</code>

Perl operators also have a specific precedence that dictates the order in which they are applied, much like the mathematical order-of-operations. The operators cited here are only the common operators available to Perl. You can find the complete list using the PerlDoc command, `perldoc perlop`.

Relational and Equality Operators

You use relational operators and equality operators in conditional tests when comparing one value to another. *Relational operators* compare two values to determine whether they are greater-than or less-than one another. *Equality operators* compare two values to determine whether they are exactly equal to one another.

There are two types of relational and equality operators, *numeric* and *string*. It is important not to confuse numeric and string operators when writing conditional tests. Use the numeric relational or equality operators when comparing two numerical values or variables.

```
if ( ( $x < $y ) && ( $y == 5 ) ) {
    # x is less than y, y is exactly 5
}
```

Use the string relational or equality operators when comparing two string values or variables.

```
if ( $text eq "" ) {
    # Text variable is an empty string
}
```

If you use a numeric operator to compare two strings, or vice-versa, you probably will not get the results you expect with the test.

String Operators

You use string operators to manipulate strings using either concatenation or repetition. You will often use the *concatenation string operator* when printing text, or appending data to an existing string. Represented as a single period, when it is placed between two values, Perl concatenates them together into a single string.

```
print "Coordinates: " . $x . ", " . $y . "\n";
```

You use the *repetition string operator* to copy and repeat a string. Represented as the letter `x`, it is most often used to format lines and spaces.

```
print "-" x 80;      # print 80 dashes.
```

Logical Operators

You use logical operators to perform logical comparisons between each side of the operator. The most common use of these operators is within multiple comparison tests, but you may also use them when assigning variables, or executing normal code.

LOGICAL OPERATOR	DESCRIPTION
&&	If left side is true, evaluate right side.
	If left side is false, evaluate right side.
//	If left side is not defined, evaluate right side.

The last two logical operators are especially useful when ensuring that a value is properly assigned.

```
$timeout = $config{ 'timeout' } || 5;
```

In this example, the local scalar `$timeout` is set to the hash `$config{ 'timeout' }` value. However, if `$config{ 'timeout' }` is not defined or equal to zero, you assign the value 5 to the scalar. Alternatively, if you want to allow for zero as a possible timeout value, you only need to ensure that `$config{ 'timeout' }` is defined. If the hash key is not defined, you set the scalar's value to 5.

```
$timeout = $config{ 'timeout' } // 5;
```

Bitwise Operators

You use *bitwise operators* to perform bitwise comparisons on either numbers or strings. Perl treats numerical bitwise operators exactly like other programming languages; the operator compares each number as a vector of bits, not as a whole number.

BITWISE OPERATOR	DESCRIPTION
&	Bitwise AND binary comparison
	Bitwise OR binary comparison
^	Bitwise XOR binary comparison

Perl's bitwise string operators are uniquely available in Perl; they use the exact same characters as the numeric operators but compare strings. Bitwise string calculation with operations can be overly complex and are actually fairly rare in most Perl code; if you need to do bitwise operations on strings, use the `vec` function.

Regular-Expression Operators

Regular-expression operators appear to be a form of combined assigned operator, except that you use them to launch a regular-expression pattern match on the left-side variable, using the pattern on the right. There are two types of regular-expression operators. The most common is `=~`, followed by its negated counterpart, `!~`. Regular-expression operators may be provided in conditional tests to validate string matches based upon supplied patterns.

```
if ( $results =~ /error/ ) {
    # Somewhere in the $results, the word
    "error" was found.
}
```

Alternatively, to check if the word "error" was not found anywhere in `$results`, use the secondary regular-expression operator `!~` in place of `=~` to negate the test.

C

You can
that val
test retu

Contro

have completed, you may use an `else` statement as a contingency block if all earlier tests have failed.

```

if ( EXPR1 ) {
    # EXPR1 is true
}
elseif ( EXPR2 ) {
    # EXPR1 was false, but EXPR2 is true
}
elseif ( EXPRn ) {
    # EXPR1..EXPRn-1 were all false, but EXPRn
    is true
}
else {
    # EXPR1..EXPRn were all false
}

```

As Perl progresses through the various tests, as soon as it encounters one that is true, it skips all subsequent `elsif` and `else` tests.

- 2 Declare some starting variables that will be tested against.

Note: Type `perldoc -f localtime` for details on the `localtime` function.

- 3 Type `if (EXPR) {` to begin a new `if` block and press Enter.

- 4 Type `}` to close the `if` block and press Enter.

- 5 Type **elseif (*EXPR*) {** to begin another `if` block and press Enter.

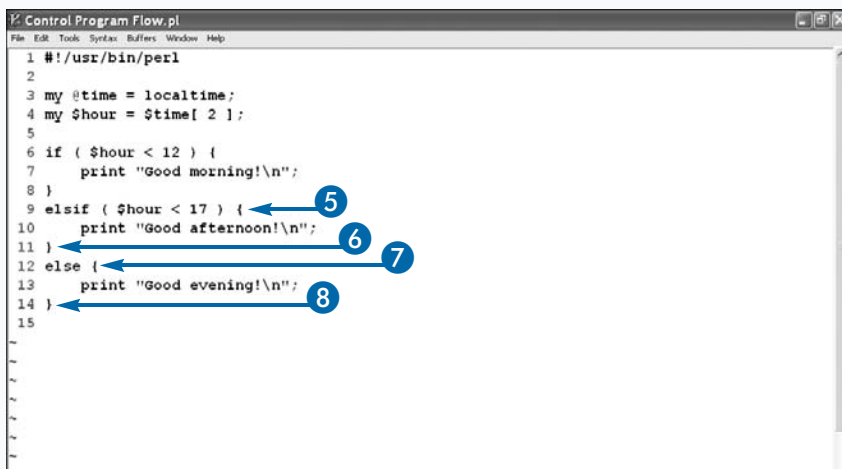
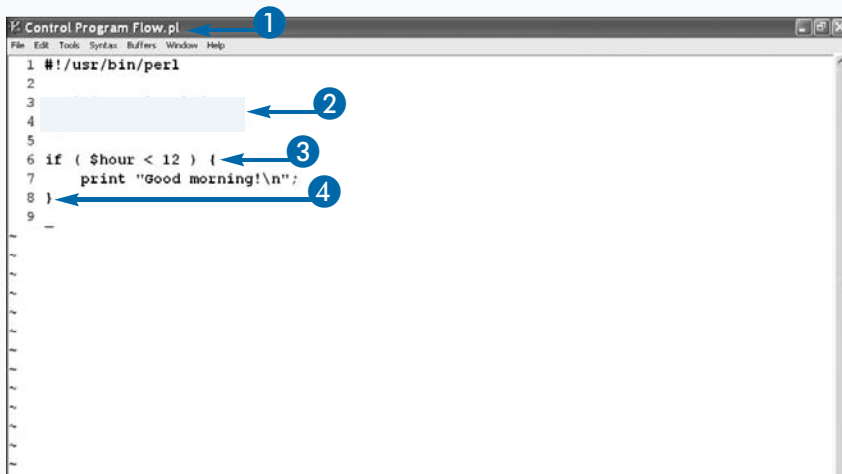
This block is only executed if all previous `if` blocks are false.

- 6 Type `}` to close the second `if` block and press Enter.

- 7 Type **else {** to create a final `else` block and press Enter.

This block is only executed if all previous `if` and `elseif` blocks are false.

- 8 Type `}` to close the final `else` block and press Enter.



- 9 Create a new `if` block nested within the first `if` block.

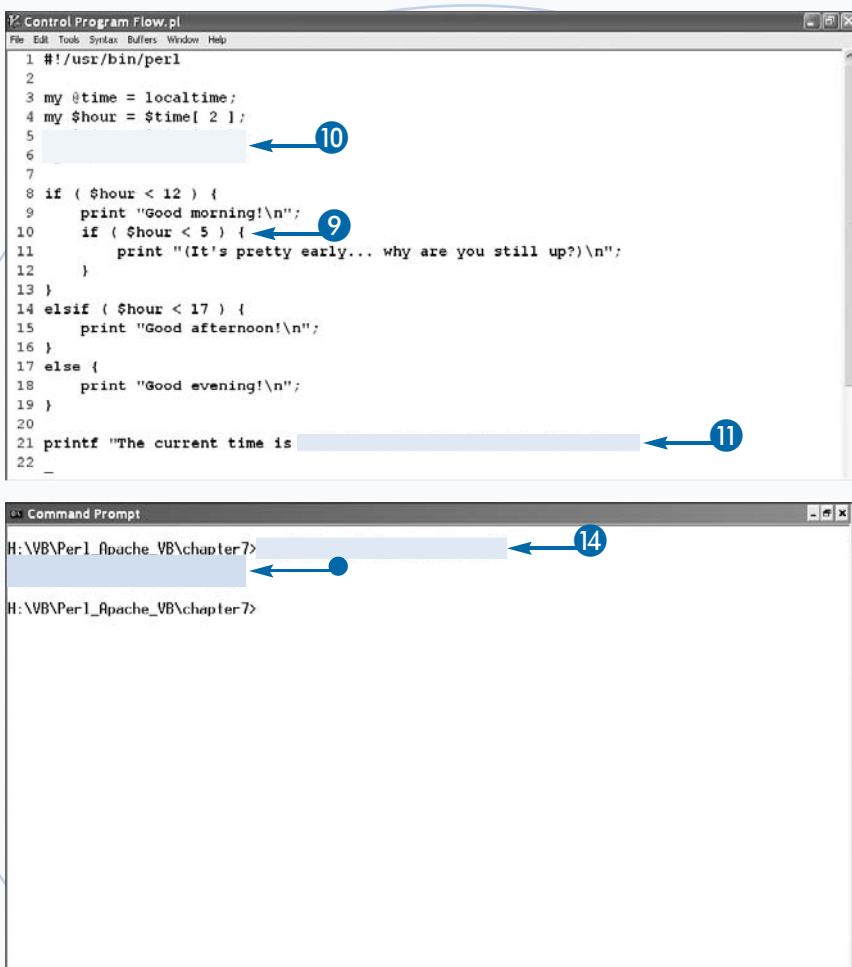
Note: Nested `if` blocks are only processed if their parent's `if` block expression is true.

- 10 Declare new variables to store the minutes and seconds.
- 11 Print to the screen the current time to validate the results.

Note: Use the `perldoc -f printf` command for details on how the `printf` function works.

- 12 Save the Perl script and exit the text editor.
- 13 Open the Command Prompt in the directory where you saved the Perl script.
- 14 Execute the script.

- The Perl script's output is displayed, evaluating the conditional tests.



Extra

As you write code, you should be asking yourself, "What happens if this function returns zero results?" or "What if the user enters bad data?" You need to be planning for the worst: give your program the capability to recover from a situation if it finds itself handling invalid results or poor data. Error correction is a technique that you must use to anticipate these problems and working around them.

Deciding what to do if problem data occurs takes some forethought. If an earlier function failed, is there an alternative that can be run instead? Could the function also return an error message and display it to the user?

It is important to note that if the programmer fails to anticipate these types of problems, and provide a solution, then unexpected program results may occur.

Sometimes a solution is to just display a vague error message and stop the program gracefully. This may seem like a lazy way to correct errors, but sometimes it is the only way. It is better for the end-user to receive an error code and report it back, than it is to have the screen suddenly turn black.

Introducing Perl Loops



Perl allows for looping code to give the developer the opportunity to address code that may occur multiple times in a row. Each pass of the loop may mean that the data being acted upon changes, but the code that reacts to each pass of data stays exactly the same.

You can use loops to read data from a file, load rows of data from a database, or even to repeatedly prompt the user to enter a correctly formatted phone number.

Identifying an exit may happen by using a conditional test on each pass; if the test fails, it exits the loop. This may also happen inherently through the looping data itself; once there is no more data to process, it exits the

loop. Perl provides two primary commands for handling conditional loops: `foreach` and `while`.

You use the `foreach` command to loop over arrays. Because an array inherently has a finite number of elements when it reaches a `foreach` loop, Perl does not require an explicit `break` statement. However, you may use `break` to prematurely exit a `foreach` loop.

You can use the `while` command to loop data that has an unknown number of entries. Often this is data that is known only outside of the Perl script, such as lines in a file, rows of a database table, or even data that the user provides.

The foreach Loop

You use the `foreach` loop to access a list of elements. A list could be a declared array variable, an anonymous array from a function (for example, keys from a hash), or a number range.

Syntax

For every element found in `LIST`, the variable `VAR` is set and `CODE` is executed.

```
foreach VAR ( LIST ) {  
    CODE;  
}
```

Alternatively, the code may precede the `foreach` statement.

```
CODE foreach VAR ( LIST );
```

If `VAR` is declared with `my` in the `foreach` loop directly, then when the loop is complete, `VAR` is no longer accessible. If `VAR` is declared before `foreach`, then its last value (the last value of `LIST`) is stored in `VAR` after `foreach` is done.

Loop an Array by Element

The simplest way to use `foreach` is to supply it with a pre-defined array.

```
my @vowels = ( 'a', 'e', 'i', 'o', 'u' );  
foreach my $letter ( @vowels ) {  
    print "$letter\n";  
}
```

The `@vowels` array has five elements; each iteration of the loop prints `$letter`, which is updated with the actual character.

Within the `foreach` block of code, it is possible to update `$letter`'s value. This in turn updates the original `@vowels` array.

The Loop Variable

For every pass `foreach` makes on the list, a variable is updated. This variable represents the specific element of the array the loop is currently working on. If no variable is provided, the special variable `$_` is implied.

Be careful, as updating `$_` is only possible when accessing the array by each element. If you are looping by the index, specify the array's scalar by index (for example, `$array[$_] = newvalue`).

Loop an Array by Index

It is possible to use the range operator `(..)` to create an anonymous array of 0 to the number of elements in the `@vowels` array. You access the length of `@vowels` using `$#` instead of `@`.

```
foreach my $i ( 0..$#vowels ) {  
    print "$vowels[ $i ]\n";  
}
```

This method of looping an array is useful if you need to access `$i`'s information separately from accessing the array itself.

```
print "$i: $vowels[ $i ]\n";
```

It is not possible to update `$i` directly. The range is provided as an anonymous array to `foreach`, `$i` is literally set to the numbers 0 to 4. Instead, you have to update the array element in the same way you access the array's individual scalar value.

The while Loop

You use the `while` loop to repeat code and run a conditional test on each pass. Typically code being repeated usually references an external source, making an unknown number of loops.

Test the Condition; Execute the Code

The standard `while` command first tests an `EXPR` condition; if true, it makes the first execution of `CODE`.

```
while ( EXPR ) {
    CODE;
}
```

The `while` command is useful when you already have data to test, but the results of that test may mean that the looping code may not be necessary.

Execute the Code; Test the Condition

You can use the `do` command to reverse `while`'s order. Instead of testing the condition first, `do` executes the first pass of code, then tests a condition and decides whether to continue.

```
do {
    CODE;
} while ( EXPR );
```

The `do` command is useful when reading files, or prompting the user with a question, or in any other scenario when you know there will be at least one iteration of code.

Stopping the Loop

The primary method to stop a `while` loop is for the `EXPR` test to fail. It is possible to control a loop's execution outside of `EXPR` by using the `next` or `last` function.

Support Functions for Loops

There are various functions that you may use to fine-tune the execution of loops in Perl.

Force the Next Loop Pass

You use the `next` command to stop processing the code block and immediately jump to the next iteration of the loop. An example of using `next` is as follows: when reading lines from a text file, if the line is a comment, skip the rest of the code in the loop with `next` and move on to the following line.

Force the Last Loop Pass

You use the `last` command to stop processing the code block and exit the loop. This has the same effect as `EXPR` failing in `while`, or when the last element of a list is reached in `foreach`.

Split a Loop into Multiple Blocks

You can use the `continue` command to split a `while` statement into multiple blocks. You can even instruct Perl to jump down or up one block using the `next` or `redo` commands, respectively.

The goto Command

When you tie the `goto` command to a `LABEL`, you can use it to provide functionality that is similar to `while` and `foreach`, but with slightly less obvious blocks of code. If a `goto` command is dozens or hundreds of lines away from its `LABEL`, then it may not be obvious to someone new to the code that this is even a loop. Avoid using `goto` unless you are already familiar with its use in other languages.

Loop Program Flow with foreach, while

Looping a program's execution flow allows the programmer to anticipate and act upon a repetitive task that may have a finite or an unknown number of iterations.

Based upon the requirements of a particular loop, two types of looping functions are available: `foreach` and `while`.

You use the `foreach` command to process a list of elements, and to execute a block of code on each element. The list may be an array variable, an anonymous array, or a range. Each element in the list is available as a scalar variable within the code itself.

```
foreach VAR ( LIST ) {  
    CODE;  
}
```

You use the `while` command to process repeating code that is usually referencing an external list, outside of the Perl interpreter. For every iteration of the loop, an expression is evaluated which allows Perl to determine whether to continue with the loop or exit.

```
while ( EXPR ) {  
    CODE;  
}
```

The expression that `while` is testing may change with every pass. It may be coming from an internal program validating a hash (with `each`), or from an external source such as a user who types something different with each pass.

Once the `LIST` ends, or `EXPR` returns false, the block is finished and normal program flow continues. In order for the loop to be useful, it should affect something declared prior to the code itself, and then act upon that event after the loop has finished.

Loop Program Flow with foreach, while

- 1 Start a new Perl script in a text editor.
- 2 Type `foreach (LIST) {` to begin a `foreach` loop block and press Enter.

Note: In this example, the list is represented as a range of numbers to loop by.

- 3 The unique value within each loop is `$_`; this represents your index.
- 4 Type `}` to close the `foreach` loop block and press Enter.
- 5 Type `while (EXPR) {` to create a `while` loop around the `foreach` loop and press Enter.
- 6 Declare a variable outside the `while` loop that will maintain the integrity of `EXPR` for the duration of the loop.
- 7 Set the variable with the raw input from `STDIN`.
- 8 Remove the trailing carriage-return from the variable.
- 9 Type `}` to close the `while` loop block and press Enter.

```
1 #!/usr/bin/perl  
2  
3 my @fields = ( "First Name", "Last Name", "Phone Number", "Address", "Country" );  
4 my @answers;  
5  
6 foreach ( 0..$#fields ) {  
7  
8  
9  
10 }  
11
```

```
1 #!/usr/bin/perl  
2  
3 my @fields = ( "First Name", "Last Name", "Phone Number", "Address", "Country" );  
4 my @answers;  
5 my $input;  
6  
7 while ( $input ne "q" ) {  
8     foreach ( 0..$#fields ) {  
9         my $field = $fields[ $_ ];  
10        my $answer = $answers[ $_ ];  
11        print "$_ $field\t: $answer\n";  
12    }  
13    print "q) quit\n";  
14    print "> ";  
15  
16    $input =   
17    chomp( $input );  
18 }  
19
```


- 10 Create an `if` block to process the input data.

Note: The regular expression `$input =~ /\d$/` validates that the user entered in exactly one digit.

- 11 Prompt the user for the answer using the same input method.
- 12 Save the Perl script and exit the text editor.

- 13 Open a Command Prompt in the directory where you saved the Perl script.

- 14 Execute the script.

Initial output of the program is displayed, and the loop begins.

- 15 Type `0` ↵.

- 16 Fill in the prompt for the first name.

The program loops, printing the results for the collected fields.

- 17 Type `1` ↵.

- 18 Fill in the prompt for the last name.

The program loops again.

- 19 Type `q` and press End to quit the program.

```

1 #!/usr/bin/perl
2
3 my @fields = ( "First Name", "Last Name", "Phone Number", "Address", "Country" );
4 my @answers;
5 my $input;
6
7 while ( $input ne "q" ) {
8     foreach ( 0..$#fields ) {
9         my $field = $fields[ $_ ];
10        my $answer = $answers[ $_ ];
11        print "$_ $field\t: $answer\n";
12    }
13    print "q) quit\n";
14    print "> ";
15
16    $input = <STDIN>;
17    chomp( $input );
18    if ( $input =~ /\d$/ ) {
19        print "Please enter your $fields[ $input ]: ";
20    }
21
22 }
23 print "\n";
24 }
25

```

```

H:\VB\Perl_Apache_VB\chapter7>
0) First Name :
1) Last Name :
2) Phone Number :
3) Address :
4) Country :
q) quit
> 0
Please enter your First Name: John
0) First Name : John
1) Last Name :
2) Phone Number :
3) Address :
4) Country :
q) quit
> 1
Please enter your Last Name: Doe
0) First Name : John
1) Last Name : Doe
2) Phone Number :
3) Address :
4) Country :
q) quit
> q
H:\VB\Perl_Apache_VB\chapter7>

```

Extra

Several alternative commands exist which you could use to rewrite the above looping code. For example, you can use `for` instead of `foreach`. However, doing so means you must replace `$_` with a real variable.

```

for ( my $i = 0; $i < $#fields; $i++ ) {
    CODE
}

```

Also, you can use `do` instead of `while`. This moves the conditional test to the end of the code, guaranteeing at least one code-block pass.

```

do {
    CODE
} while ( $input ne "q" );

```

Finally, the example using `STDIN` as a method to collect data from the user is rather crude. A better module to use is `Term::ReadLine`.

Introducing Perl Subroutines

Perl subroutines allow for the programmer to externalize code into a separate block, identify the block with a name, and execute the block at any time inside of the main program or any other subroutine.

Subroutines are also useful for reducing the amount of repeating code within a main program, and instead house that code within a common subroutine that is called when it is needed.

One or more subroutines may exist internally within a single Perl script, externally within in a shared file, or within a third-party module.

Beyond what is described here, Perl supports additional subroutine features including attributes, prototypes, and persistent local variables. For more information on these advanced features, use the `perldoc perlsub` command.

Syntax of Subroutines

Utilizing a subroutine within Perl is relatively simple; however, you need to pay special attention when passing parameters to it and accepting return values from it, and you need to understand that the code block within a subroutine is a stand-alone block when compared to the rest of the program.

Define a Subroutine

You can define subroutines anywhere within a Perl script. This means they may occur anywhere before or after where they are actually called. The syntax for a subroutine always begins with `sub`, followed by the subroutine's name, followed by a code block.

```
sub subroutine { CODE }
```

A subroutine's name is case-sensitive.

Execute a Subroutine

Once you declare the subroutine, you can call it using the syntax

```
&subroutine( PARAMS );
```

If no parameters are required, leave the brackets in place when calling the subroutine

```
&subroutine();
```

Perl allows you to execute a subroutine without an ampersand or parenthesis, but this results in implicitly passing parameters from one subroutine to another. Generally, you should avoid this unless you know what you are doing and how the subroutine should act. You can find more information in the `perlsub` manual.

Pass Parameters to a Subroutine

Parameters need to be received using the special array, `@_`. A subroutine can receive its parameters in two ways, all at once,

```
sub subroutine {  
    my ( VAR1, VAR2, ..., VARn ) = @_;  
    CODE  
}
```

or one parameter at a time using `shift`. Note that the `shift` function normally accepts an array as a parameter. If an array is not supplied, `@_` is implicitly used.

```
sub subroutine {  
    my VAR1 = shift;  
    my VAR2 = shift;  
    ...;  
    my VARn = shift;  
    CODE  
}
```

It is important to note that passing a parameter into a subroutine is the equivalent of assigning a value to a new variable. Any modifications to the new variable within the subroutine have no effect on the original parameter value outside the scope of the new subroutine.

If the purpose of a subroutine is to modify the parameter being passed, and to maintain that modification for the scope of the calling code block, then you need to use a reference variable. For more information on references, see Chapter 8.

Return Results from a Subroutine

It is good practice for a subroutine to return results at the end of its code block. This provides an easy way to communicate information back to the code block that called the subroutine without using references.

```
VAR = &subroutine();
sub subroutine {
    CODE
    return VALUE;
}
```

The `return` function immediately stops the subroutine's execution and sends `VALUE` to `VAR`. Any code below it is ignored. It is legal for `return` to exist within a conditional block, resulting in controlled execution of a subroutine's components based upon conditional tests.

If a subroutine is going to return multiple return values, then you pass an array through `return`.

```
( VAR1, VAR2, ..., VARn ) = &subroutine();
sub subroutine {
    CODE
    return ( VALUE1, VALUE2, ..., VALUEn );
}
```

Try to avoid this method of passing multiple values back to the calling code block. It is very easy to lose track of how many values are being returned, and where they are within the array. Also, if five `VALUES` are returned and assigned into six `VARS`, only the first five receive a value. The sixth is undefined.

Because `return` natively handles arrays, passing a single array from a subroutine to a parent code block is perfectly acceptable.

```
ARRAY = &subroutine();
sub subroutine {
    CODE
    return LIST;
}
```

Using a Shared Subroutines File

It is possible to externalize a subroutine into a separate file, and include this file at run-time in a main script. This has the benefit of simplifying a larger main program, and of sharing subroutines among multiple main programs. However, this method is viewed as being ad hoc and should not be used for large, long-term program development; create a Perl module instead.

Create a Shared Subroutine File

To create a shared subroutine file, create a new file and start defining only subroutines. The filename should have the extension `.pl`, just like a regular Perl script; however, the file must return a *true value* on its last line. This is necessary for the `require` function to import the file correctly. For example, a shared subroutine file, `SHAREDFILE.pl`, may simply end with `1;`.

```
sub subroutine1 { CODE }
sub subroutine2 { CODE }
...
sub subroutineN { CODE }
1;
```

Import a Shared Subroutine File

Importing a shared subroutine file is handled by the `require` function. You should insert this function fairly close to the beginning of all Perl scripts that need it.

```
#!/usr/bin/perl
require "SHAREDFILE.pl";
&subroutine1();
&subroutine2();
...
```

Using Perl Modules

You use Perl modules for generalizing complex Perl code into a simple interface. You can define and execute a set of common subroutines through a Perl module, thus simplifying the code complexity of the main program. You can find more information about Perl modules, including creating, downloading, and installing them, in Chapters 8 and 9.

Organize Program Code with Subroutines

Intelligently organizing program code in subroutines allows a programmer the flexibility to simplify the code, which makes debugging, building, and maintaining long-term code development easier. You can use subroutines to initialize global variables, reset states, load and save data, display controlled messages — literally anything.

Identifying whether to use a subroutine is a creative judgment call of the developer. Ask yourself, “Who will be viewing my code? Is its function and purpose obvious? What happens if a problem occurs in a subroutine? Can I recover the situation?”

It is a good idea to use a subroutine to provide access to code that handles complex calculations, even if you only use it once. A function with an appropriate name, such as `CalculateUserScore`, is very straightforward.

Organize Program Code with Subroutines

Sometimes, when reviewing code that you wrote months or even years ago, an appropriate naming convention for subroutines can be enough to jog your memory on how your own program works!

You should use comments with subroutines, not only within the contained code block, but also to introduce the subroutine itself. Provide details such as the summary of the subroutine, and explain what it does and the intended end result. Summarize the subroutine’s input variables, and return values and any global variables it may modify, as well as its raw output, if applicable.

Remember that subroutines are infinitely flexible. A single subroutine can reference other subroutines, even itself, in its own code. A subroutine may contain conditional tests, looping code, or simply a group of statements that displays static data to the screen.

1 Open a Perl script in a text editor.

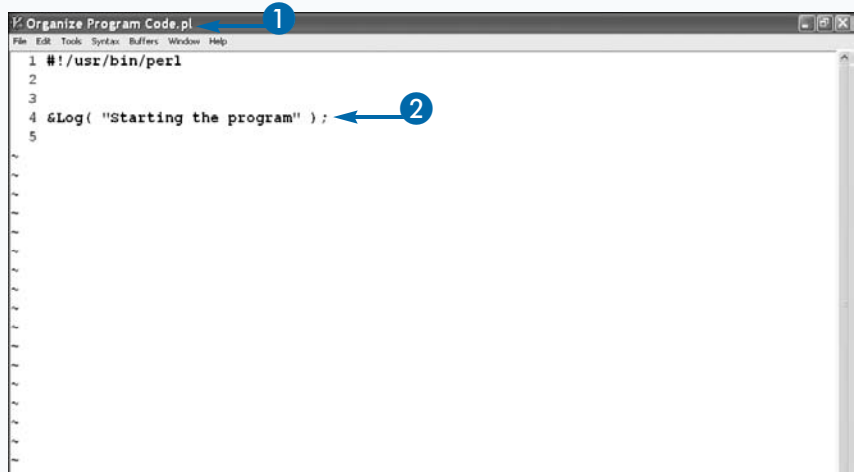
2 Type `&subroutine(TEXT);` to call the subroutine with the text as a parameter, and press Enter.

3 Insert comments that introduce the subroutine’s name and parameters.

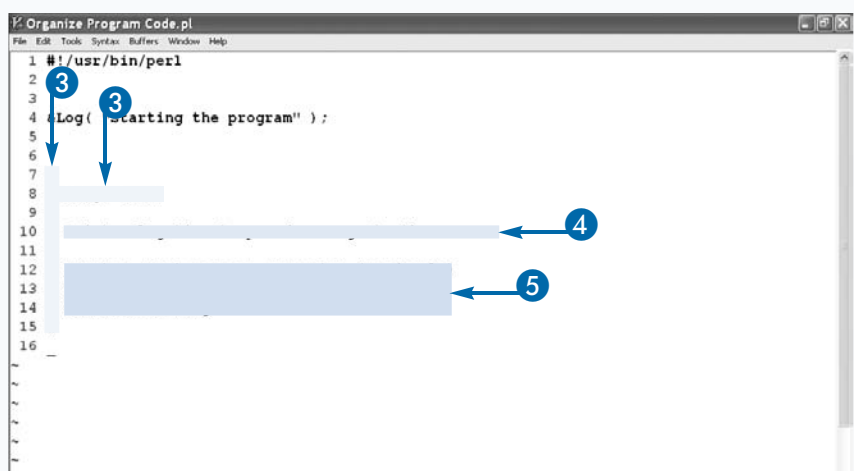
4 Type a description for the subroutine.

5 Summarize the subroutine’s expected input, output, and return values.

Note: Input does not always refer to input parameters. It could also be global variables, an external file, a user typing at the keyboard, or even another subroutine’s results.



```
1 #!/usr/bin/perl
2
3
4 &Log( "Starting the program" );
5
```



```
1 #!/usr/bin/perl
2
3
4 &Log( "Starting the program" );
5
6
7
8
9
10
11
12
13
14
15
16
```

- 6 Type **sub subroutine {** to begin the subroutine block and press Enter.
- 7 Type **my VAR = shift;** to read the first parameter as a newly declared variable.
- 8 Type **}** to complete the subroutine code block.
- 9 Add more `&subroutine()` functions.
- 10 Save the Perl script and exit the text editor.
- 11 Open a Command Prompt in the directory where you saved the Perl script.
- 12 Execute the Perl script.
 - Output of the Perl script is displayed using subroutines.

```

1 #!/usr/bin/perl
2
3
4 &Log( "Starting the program" );
5
6 #
7 #
8 # &Log( $VAR )
9 #
10 # Print a log timestamp and message to the screen.
11 #
12 # Inputs:  Text message parameter (read only)
13 # Outputs: Print the time and a message
14 # Returns: nothing
15 #
16 {
17     my $message = shift;
18     my $time = localtime;
19     print "$time -- $message\n";
20 }
21
22 my $x = 5;
23 my $y = 10;
24 &Log( "Initialized variables: x = $x, y = $y" );
25
26 &Log( "Program completed" );
27
28
29
30

```

```

H:\VB\Perl_Apache_VB\chapter>perl "Organize Program Code.pl"

```

Apply It

Perl makes some assumptions about how certain functions work, especially if it appears as though incorrect or incomplete data was supplied. The use of the `shift` function in subroutines is an example of this assumption. The syntax for `shift` is `VAR = shift(ARRAY)`. Because you are using `shift` without an `ARRAY`, it will default to the subroutine's special array of input parameters, `@_`. If you are passing multiple parameters, you can use multiple calls to `shift`, one per line, and load the retrieved parameter data into variables.

```

my VAR1 = shift;
my VAR2 = shift;
...
my VARn = shift;

```

If you are passing a lot of parameters, that can take up a lot of space. Instead, you can access `@_` directly with the syntax: `my (VAR1, VAR2, ..., VARn) = @_`. Even if there is only one variable being loaded, this format works. To put it in the context of the lesson, step 7 could be rewritten as: `my ($msg) = @_`. Remember, when reading data from an array as an array, there must still be brackets around the variables being populated, even if there is only one variable.

Manipulate Variables in Subroutines

Manipulating the state of a program in mid-execution can make the program respond to the user, react to its surroundings, and perform a specific function that may not occur in normal program execution is very important. You use variables to keep track of various states throughout a program's lifetime. You have seen that updating a variable in the main program is rather easy; if the variable is global to the script, then an update within a subroutine is exactly the same.

However, if a local variable crosses into a subroutine, and that subroutine is tasked with updating its value, then you need to make special considerations.

Simply supplying a variable as an input parameter into a subroutine will make its value available in the new code

block, but if that variable is going to change, and the original caller needs the new value, then the variable should be changed into a reference.

An alternative to using a reference is to use a *global variable*. A global variable does not need to be passed as an input parameter into a subroutine; it can even be read and updated just like in the main program. However, global variables can become difficult to maintain and keep track of as a program becomes larger.

Ideally, you should only make a variable global if you legitimately need it to be available to every subroutine in a program. If that is not the case, and you still require a subroutine to update a variable, use its reference.

Manipulate Variables in Subroutines

- 1 Open a Perl script in a text editor.
 - 2 Declare a globally-scoped variable.
 - 3 Declare a locally-scoped variable inside a subroutine.
- Note:** See Chapter 6 “User-Defined Variables” for more information about globally- and locally-scoped variables.
- 4 Manipulate both variables within another subroutine.
 - 5 Print both variables to review the results.
 - 6 Save the Perl script.

- 7 Open a Command Prompt in the directory where you saved the Perl script.
- 8 Execute the script.
 - The globally-scoped variable is modified.
 - The locally-scoped variable is not modified.

```
1 #!/usr/bin/perl
2
3
4
5
6 # Begin main program
7 $main();
8
9 sub main
10 {
11     [redacted]
12
13     $AddToVar( 3 );
14
15     [redacted]
16
17 }
18
19
20 sub AddToVar
21 {
22     my $value = shift;
23
24     [redacted]
25
26 }
27
```

```
H:\VB\Perl_1\Apache_VB\chapter7> perl ManipulateVariables.pl
globalVar: 3
localVar: 0
H:\VB\Perl_1\Apache_VB\chapter7>
```


- 9 Press Alt+TAB to switch back to the text editor.
- 10 Type `\$scalar` to supply a reference of the local variable to the subroutine.
- 11 Add a new parameter to the subroutine to accept a referenced local variable.
- 12 Type `$$scalarref` to access the dereferenced variable value.

Note: This example demonstrates referencing and dereferencing a scalar. See Chapter 8 for examples using arrays and hashes.

- 13 Save the file and exit the text editor.
- 14 Press Alt+Tab to switch to the Command Prompt.
- 15 Execute the script.
 - The globally-scoped variable is modified.
 - The locally-scoped variable is modified.

The top screenshot shows a Perl script editor titled "Manipulate Variables in Subroutines.pl". The script contains the following code:

```

1 #!/usr/bin/perl
2
3 # Initialize globally-scoped variables
4 my $globalVar = 0;
5
6 # Begin main program
7 &main();
8
9 sub main
10 {
11     # Initialize locally-scoped variables
12     my $localVar = 0;
13
14     &addToVar( 3, <-- 9
15
16     print "globalVar: $globalVar\n";
17     print "localVar: $localVar\n";
18 }
19
20 sub addToVar
21 {
22     my $value = shift;
23     my $localVarRef = shift; <-- 10
24
25     $globalVar += $value;
26     $localVarRef += $value;
27 }
28
29
30

```

Annotations 9, 10, and 11 point to the arguments in the `&addToVar` call on line 14. Annotation 9 points to the value `3`, annotation 10 points to the reference `&`, and annotation 11 points to the subroutine name `addToVar`.

The bottom screenshot shows a Command Prompt window. The output of the script is:

```

H:\VB\Perl_Apache_VB\chapter7>perl "Manipulate Variables in Subroutines.pl"
globalVar: 3
localVar: 0
H:\VB\Perl_Apache_VB\chapter7>
globalVar: 3
localVar: 3
H:\VB\Perl_Apache_VB\chapter7>

```

Annotation 14 points to the output of the script, and annotation 15 points to the execution command.

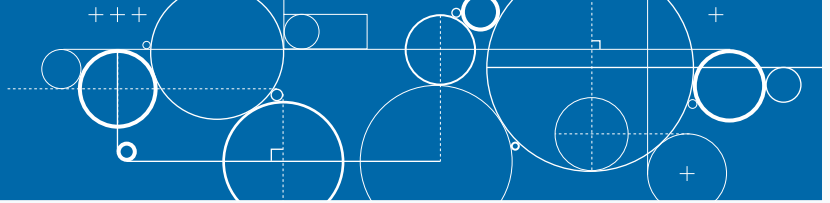
Extra

When using references in subroutines, think of them like *pointers* in C and C++. A reference variable does not contain the actual data, but the memory address of the data. When passing data to a subroutine, and the data is to be updated, a reference is always preferred.

The naming convention here is extremely important. To keep track of everything, if you receive a scalar reference as input, give it a name to imply that it is a reference: `$localVarRef`. By ending it in "Ref", you now know that attempts to access that original data must be preceded by two dollar-signs, not one.

The concept of references does become simpler in the context of arrays and hashes compared to this example and scalar references. This is because you must treat complex nested variables as references, in order to keep track of data in each nested dimension. For more information, see Chapter 8.

Introducing References



Perl references act as pointers to information that is stored somewhere else. The idea is based upon C or C++ pointers, except that Perl maintains the structure of the referenced data in its original form.

You normally use a scalar variable to store the lowest level of a reference in the Perl code that originally initialized the reference. In fact, when a Perl module is initialized, its handle is made available as a scalar. From

that scalar reference, all of the module's variables, libraries, and methods are made available to the original Perl script.

For subroutines, using a reference as a parameter makes it easier to update the data within the subroutine itself.

References are actually the precursor to Object-Oriented programming in Perl. For more information on references, run the PerlDoc command `perldoc perlref`.

The Scalar Reference

A scalar reference has two basic functions in Perl. First, you can use it to provide a reference to a scalar. Second, a scalar reference is actually just a normal scalar holding a reference to a more complex data structure, such as an array, hash, or subroutine directly, as described in this section.

Creating a Scalar Reference

You can create a scalar reference using a simple backslash in front of a predeclared scalar variable.

```
$scalarref = \$scalar;
```

Now, `$scalarref` may be passed as a parameter into a subroutine.

Dereferencing a Scalar Reference

You can dereference a scalar reference to access its original data. Since a reference is just like a pointer in C and C++, dereferencing a variable in Perl is just like following the pointer to the actual data, except in the context of a Perl scalar.

In the context of a subroutine that has accepted a scalar reference as a parameter, the original value can be dereferenced, read, and updated using two dollar-signs.

```
$scalar = $$scalarref;
```

In the context of a subroutine that has accepted a scalar reference as a parameter, the new value is automatically applied to the original scalar variable that you created with the backslash.

```
print $$scalarref;
$$scalarref = VAR;
```

The Array Reference

An array reference is a pointer to an array that is stored within a scalar. The individual scalar can then be passed from one program component to another, and the original array's contents remain accessible.

Creating an Array Reference

You can create an array reference using a simple backslash in front of a predeclared array variable.

```
$arrayref = \@array;
```

You can create an *anonymous array* reference can be created using square brackets around a list of variables. An

anonymous array reference is just like array reference, but it is built without using a regular array variable to house the values.

```
$arrayref = [ VAR1, VAR2, ..., VARn ];
```

In both cases, the actual variable that houses the referenced data is a scalar. Now `$arrayref` can be passed throughout a program and its contents remain accessible in memory.

The Array Reference *(continued)*

Following an Array Reference

In order to access the contents of the array reference, use two dollar-signs and treat it like a normal array. Alternatively, use an arrow, `->`, between the reference and the square brackets and index. In effect, accessing and updating an array reference is exactly the same as with a normal array, except for the arrow or dollar sign.

Dereferencing an Array Reference

You would dereference an array reference to access the original data in the context of an array. When used with an assignment operator, the contents of a dereferenced array reference are copied into a new array variable.

To identify the length of a dereferenced array reference, replace `@$` with `$#`. Again, this is just like a normal array except for the extra dollar sign.

The Hash Reference

A hash reference is a pointer to a hash that is stored within a scalar. The individual scalar can then be passed from one program component to another, and the original hash's contents remain accessible.

Creating a Hash Reference

You can create a hash reference using a simple backslash in front of a predeclared hash variable.

```
$hashref = \%hash;
```

You can create an *anonymous hash* reference using curly brackets around a group of key/value pairs. An anonymous hash reference is just like hash reference, but it is built without using a regular hash variable.

```
$hashref = {
    KEY1 => VAR1,
    KEY2 => VAR2,
    ...,
    KEYn => VARn
};
```

In both cases, the actual variable that houses the referenced data is a scalar.

Following a Hash Reference

In order to access the contents of the hash reference, use two dollar-signs and treat it like a normal hash. Alternatively, use an arrow, `->`, between the reference and the curly brackets and key. In effect, accessing and updating a hash reference is exactly the same as with a normal hash, except for the arrow.

Dereferencing a Hash Reference

You would dereference a hash reference to access the original data in the context of a hash. When used with an assignment operator, the contents of a dereferenced hash reference are copied into a new hash variable. You will need to dereference hash references when dealing with built-in Perl functions that interact with hashes, but not hash references.

The Code Reference

A code reference is a pointer to a subroutine that is stored within a scalar. The individual scalar can be passed from one program component to another, making original subroutine's code accessible everywhere the scalar is in scope.

Creating a Code Reference

You can create a code reference using a simple backslash in front of an existing subroutine.

```
$coderef = \&subroutine;
```

You can create an *anonymous code* reference using the keyword `sub`, followed by curly brackets representing a new code block. An anonymous code reference is just like code reference, but it is built without using defining a regular subroutine.

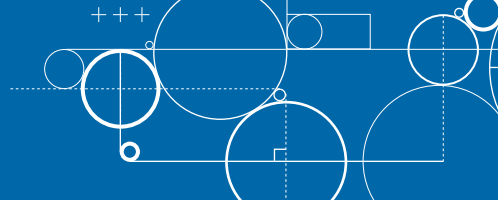
```
$coderef = sub {
    CODE
};
```

Executing a Code Reference

Code references are not exactly dereferenced; they are simply executed, just like a regular subroutine.

```
&$coderef( PARAM1, PARAM2, ..., PARAMn );
```

Understanding Compound Data Structures



Compound data structures within Perl references allow for complex data schemas that contain a mix of scalars, arrays, and hashes. Any scalar variable can hold a reference to another object, such as another scalar, an array, or a hash. Because arrays hold lists of scalars, and hashes hold tables of scalars, it is now possible to build complex structures such as arrays of arrays, arrays of hashes, hashes of arrays, or hashes of hashes, up to an infinite depth.

It is also possible to induce a *recursive loop* with compound data structures. This means that a value stored deep within the compound data structure actually references any other reference, including its own parent, grandparent, or even the lowest-level scalar that references the entire object!

The concept of compound data structures will become very important later when dealing with Perl modules, especially with the `HTML::Template` module. For more information, see Chapter 13.

Nesting References

Nesting references together is a technique designed to build complex data structures as multiple dimensions of data. In other words, an individual array or a hash contains one dimension of data. When one of those data values contains another array or hash, you now have two dimensions of data. The process can repeat itself endlessly.

If that array or hash is converted into an array or hash reference, along with all subsequent dimensions, the steps required to keep track of the data are simplified to the point that all you need is the parent reference. In other words, if you pass the parent reference to a subroutine, or to `Data::Dumper`, or anything, then all other dimensions remain easily accessible.

The Data::Dumper Module

The `Data::Dumper` module is extremely useful. It allows you to examine the contents of any type of variable, up to an infinite depth. Because references may contain any number of scalars, arrays, hashes, or other references, it is possible to lose track of what data exists where. `Data::Dumper` allows you to review the contents of any type of data object at any point in a Perl script. To use the module, first initialize it at the top of a Perl script.

```
use Data::Dumper;
```

Then, anywhere you need to review the contents of a variable, call the `Dumper` function with the variable you want to examine.

```
print Dumper( VAR );
```

For more information on how to use this module, run the Perl Documentation command, `perldoc Data::Dumper`.

Synonyms in Syntax Variation

When writing code that handles references, data housed in references, or functions that use data housed in references, Perl allows for a few syntax synonyms. As described in the earlier sections, “Following an Array Reference” and “Following a Hash Reference,” it is possible to interchange a double dollar-sign or an arrow, but things become complicated when dealing with multi-dimensional methods. Each method describes the same thing, but they are written in a slightly different way.

Synonyms in Syntax Variation *(continued)***Following a Multi-Dimensional Reference**

Following a reference allows you to gain access to a value at a specific location. When dealing with a multi-dimensional reference, all methods work, but it is important to keep everything concise and clear.

Method #1 (Weak)

Dereferencing the reference with a double dollar-sign may seem like a good idea; for example, it works for individual `scalarrefs`,

```
print $$ref{ KEY }{ KEY }{ KEY };
```

However, this method falls apart when dealing with the multiple dimensions of keys. If you want to stop processing the reference at a specific location, say at the second dimension, and then continue using another format, the syntax becomes unnecessarily complex.

Method #2 (Good)

Following the C and C++ syntax for dereferencing a pointer, the arrow (`->`) acts as a visual queue to follow the reference to its memory location.

```
print $ref->{ KEY }{ KEY }{ KEY };
```

The only problem with this method is that the subsequent keys may or may not be references. Perl makes an attempt to guess, but if it guesses wrong, your expected data will not be accessible.

Method #3 (Best)

Placing arrows across each reference hop is the best way to instruct Perl (and anyone else reading your code) that this hash reference has multiple dimensions of hash references.

```
print $ref->{ KEY }->{ KEY }->{ KEY };
```

This method is best because it is clear that you are following a reference from start to finish, or from parent scalar reference to key to reference to key to reference to key.

Dereferencing a Multi-Dimensional Reference

Dereferencing a reference allows you to gain access to the original constructor, which is a scalar, an array, or a hash, at a specific point in a multi-dimensional reference. When dealing with a multi-dimensional reference, not all methods work as expected.

Method #1 (Weak)

```
%hash = %$ref;
%hash = %$ref{ KEY };
%hash = %$ref{ KEY }{ KEY };
```

Only when dereferencing the top-level hash reference, as shown on the first line, does this method work. The next two examples fail with a syntax error because Perl is confused about which hash you are referring to.

Method #2 (Good)

```
%hash = %{ $ref };
%hash = %{ $ref->{ KEY } };
%hash = %{ $ref->{ KEY }{ KEY } };
```

By wrapping the entire reference within curly brackets *before* dereferencing, and then prefixing the block with a percent sign, you clearly instruct Perl on what you are trying to do. This method works regardless of the depth.

Method #3 (Best)

It is also acceptable, but not required, to put arrows across each additional reference.

```
%hash = %{ $ref };
%hash = %{ $ref->{ KEY } };
%hash = %{ $ref->{ KEY }->{ KEY } };
```

This method is best because it is clear that you are dereferencing a hash by jumping from scalar to reference to key to reference to key.

Build an Array or Hash Reference

Building an array or hash reference allows you to construct a standard array or hash but use a single scalar to store and reference its data. This gives you the benefit of only needing to refer the top-level scalar in your code, when you intend to access the entire array or hash.

It is best to think of an array or hash reference as a pointer to an array or hash structure in memory. If you were to examine a referenced variable and print it like a regular scalar, you would see that it holds what appears to be a memory address, like "ARRAY(0x892f880)". If you see this when you expect to see normal data, or vice versa, you are accessing a referenced variable incorrectly. The Data::Dumper module is a great way to examine a reference's data structure, incase you are unsure about what information a particular reference actually holds.

Building an array or hash reference is relatively simple; they can be from an existing array or hash variable, or from an anonymous array or hash. The difference between the two methods is that the latter does not require a separate array or hash variable. A list of multiple key/value pairs is converted into a reference using square brackets and curly brackets, respectively. Both methods are demonstrated in this section.

You can manipulate data in an array or hash reference using the same tools that you use to manipulate an array or hash, but only after the variable has been dereferenced. By mastering the idea behind an array or hash reference, you pave the way for nesting complex variable types, which gives you the ability to construct complex data structures.

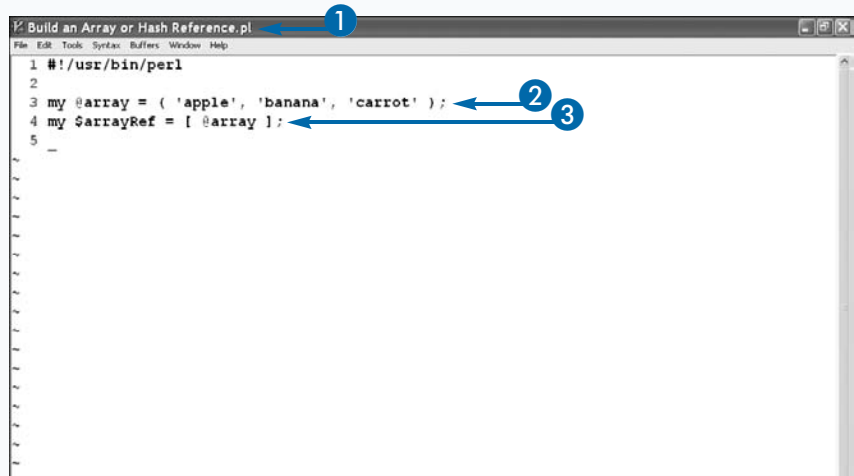
Build an Array or Hash Reference

- 1 Open a Perl script in your editor.
- 2 Type `my @array = (LIST);` to create a normal array.
- 3 Type `my $arrayRef = [@array];` to convert the normal array into an array reference.

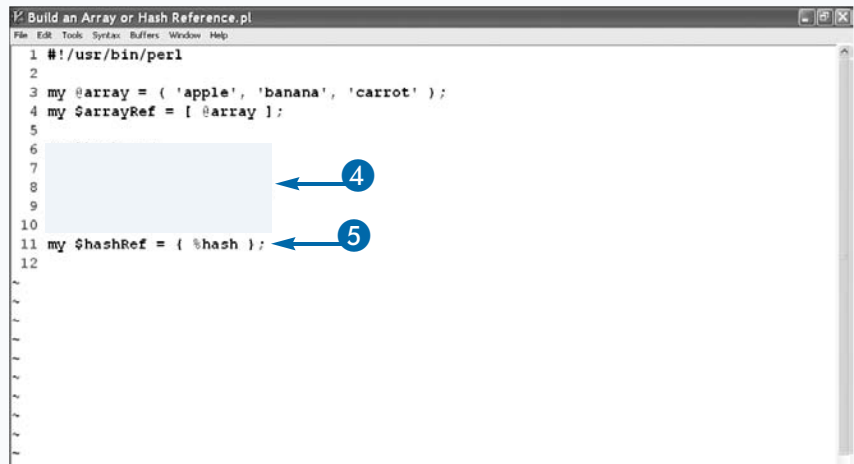
Note: Instead of using square brackets, it is legal to use a backslash. For example: `my $arrayRef = \@array;`

- 4 Type `my %hash = (KEY => VAL, ...);` to create a normal hash variable.
- 5 Type `my $hashRef = { %hash };` to convert the normal hash into a hash reference.

Note: Instead of using curly brackets, it is possible to use a backslash. For example: `my $hashRef = \%hash;`



```
1 #!/usr/bin/perl
2
3 my @array = ( 'apple', 'banana', 'carrot' );
4 my $arrayRef = [ @array ];
5
```



```
1 #!/usr/bin/perl
2
3 my @array = ( 'apple', 'banana', 'carrot' );
4 my $arrayRef = [ @array ];
5
6
7
8
9
10
11 my $hashRef = { %hash };
12
```


- 6 Type **my \$anonArrayRef = [LIST]**; to create an anonymous array reference.
- 7 Type **my \$anonHashRef = { KEY => VAL }**; to create an anonymous hash reference.
- 8 Type **use Data::Dumper**; to import the Data::Dumper module into the script.
- 9 Type **print Dumper(LIST)**; to display the contents of all array and hash references.
- 10 Save the Perl script.
- 11 Open a Command Prompt in the same directory where you saved the Perl script.

- 12 Execute the Perl script.

Several array and hash references are constructed, then displayed on-screen.

- \$VAR1 and \$VAR2 represent the first two variables sent to the Dumper function.
- \$VAR3 and \$VAR4 represent the last two variables sent to the Dumper function.

```

1 #!/usr/bin/perl
2
3 use Data::Dumper;
4
5 my @array = ( 'apple', 'banana', 'carrot' );
6 my $arrayRef = [ @array ];
7 my $anonArrayRef = [ 'apple', 'banana', 'carrot' ];
8
9 my %hash = (
10     'apple' => 'red',
11     'banana' => 'yellow',
12     'carrot' => 'orange'
13 );
14 my $hashRef = { %hash };
15
16
17
18
19
20
21 print Dumper( $arrayRef, $anonArrayRef, $hashRef, $anonHashRef );
22

```

```

H:\VB\Perl_Apache_VB\chapter8>perl "Build an Array or Hash Reference.pl"
$VAR1 = [
  'apple',
  'banana',
  'carrot'
];
$VAR2 = [
  'apple',
  'banana',
  'carrot'
];
$VAR3 = {
  'apple' => 'red',
  'banana' => 'yellow',
  'carrot' => 'orange'
};
$VAR4 = {
  'apple' => 'red',
  'banana' => 'yellow',
  'carrot' => 'orange'
};

```

Extra

To temporarily dereference the array reference first, use `@{ $arrayref }`. Now use it like a normal array.

```

push( @{ $arrayref }, LIST ); # Append data to the array reference
$value = pop( @{ $arrayref } ); # Retrieve and remove the last value

```

When using various built-in functions that require a normal hash, such as `keys` or `each`, you still need to temporarily dereference the hash reference first, using `%{ $hashref }`, and treat it like a normal hash.

```

@allkeys = keys( %{ $hashref } );
@allvalues = values( %{ $hashref } );

```

Retrieving and updating data in an array or hash reference is just like an array or hash, except you use a slightly different syntax with the arrow (`->`) to follow the reference to the actual data location in memory.

```

$arrayref->[ INDEX ] = VALUE; # Update the arrayref's value at INDEX
$hashref->{ KEY } = VALUE; # Update the hashref's KEY with VALUE

```

Deconstruct a Reference

It is sometimes necessary to deconstruct an array or hash reference in order to gain access to the data within when functions or subroutines are expecting a normal array or hash variable. It is possible to *temporarily deconstruct* the reference by converting it into its primitive form within the context of a function's parameter. This provides the benefit of allowing the function access to update the referenced data without actually needing an intermediary array or hash variable.

It is also possible to *permanently deconstruct* the reference by converting it into its primitive form as a new array or hash variable. This method is not always recommended, as any updates to the new array or hash will not update the original array or hash reference. By permanently deconstructing a reference, you are effectively copying the referenced data into a new variable.

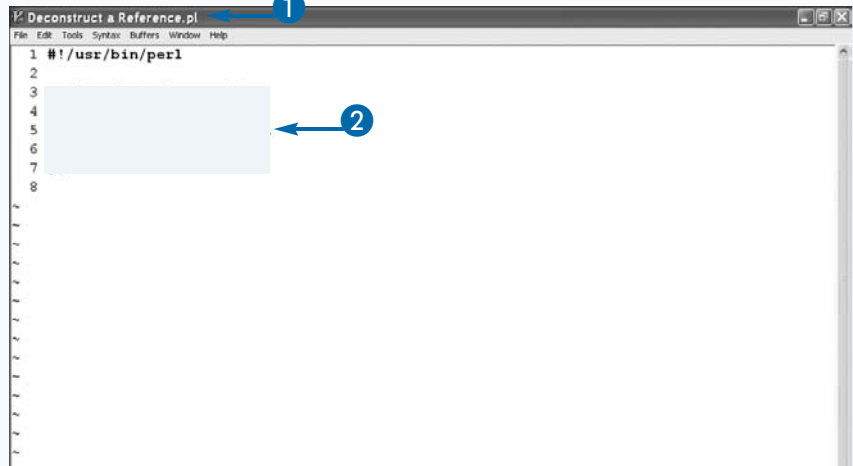
Deconstruct a Reference

Deciding which method to use to deconstruct a reference depends on how you are going to use it. Are you planning on changing something that affects the original data reference? Or, are you looking at extracting the data in the reference and using it in a new form or idea, but keeping the original reference data intact?

When dealing with referenced data within subroutines, do not feel obliged to deconstruct a referenced variable just to gain access to its data within the context of a subroutine. Remember, the whole point behind references is to pass a pointer to the data, not the actual data.

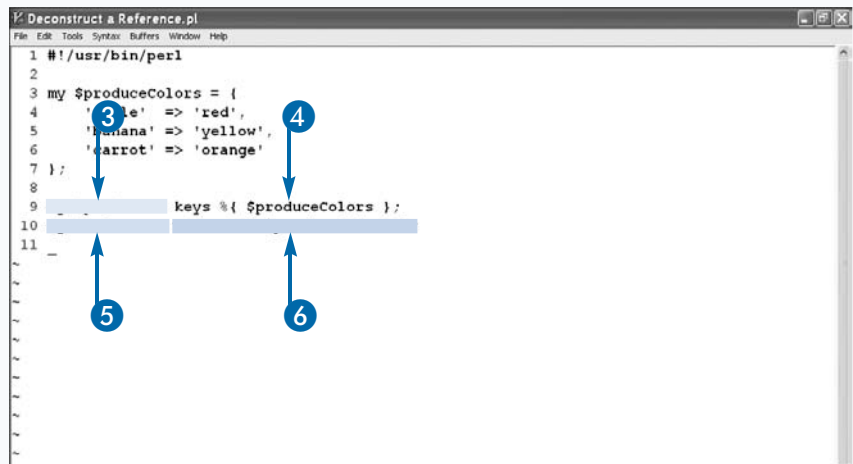
You only need deconstruct a reference immediately prior to calling a subroutine that has no concept on how to handle references. This is usually true for built-in Perl functions that are designed to handle arrays or hashes directly.

- 1 Open a Perl script in your editor.
- 2 Type **my \$produceList = { KEY => VAL, ... };** to anonymously create a new hash reference of produce.



```
1 #!/usr/bin/perl
2
3
4
5
6
7
8
```

- 3 Type **my @produce =** to create a new array.
- 4 Type **keys %{\$produceColors};** to deconstruct the hash reference and return an array of its keys.
- 5 Type **my @colors =** to create another new array.
- 6 Type **values %{\$produceColors};** to deconstruct the hash reference and return an array of its values.



```
1 #!/usr/bin/perl
2
3 my @produce =
4     'apple' => 'red',
5     'banana' => 'yellow',
6     'carrot' => 'orange'
7 };
8
9 keys %{$produceColors};
10
11 my @colors =
12     values %{$produceColors};
13
```

Note: The *keys* and *values* functions convert a hash into an array of its keys and values, respectively.

- 7 Type **sort** before keys and values.

Note: The **sort** function sorts arrays alphabetically.

- 8 To print the output of each array, type **join(' ', ARRAY)** to convert the array into a string.

- 9 Save the Perl script.

- 10 Open a Command Prompt in the same directory where you saved the Perl script.

- 11 Execute the Perl script.

- Output of the program.

The hash reference is deconstructed into new array variables, which are printed as a string.

Note: This example permanently deconstructs the original reference, making a copy of the data. By sorting its output, *apple* and *orange* are the first elements in their respective arrays, but this has no effect on an *apple* being *red* as described in `$produceColors`.

```

1 #!/usr/bin/perl
2
3 my $produceColors = {
4   'apple' => 'red',
5   'banana' => 'yellow',
6   'carrot' => 'orange'
7 };
8
9 my @produce = keys %{$produceColors};
10 my @colors = values %{$produceColors};
11
12 print "Sorted list of all produce: " . join( ' ', @produce ) . "\n";
13 print "Sorted list of all colors: " . join( ' ', @colors ) . "\n";
14

```

```

H:\VB\Perl_Apache_VB\chapter8>
H:\VB\Perl_Apache_VB\chapter8>

```

Apply It

You can deconstruct a reference only when passing data to a Perl function that expects an array or hash, not an array or hash reference. To create a subroutine that accepts a referenced variable, treat its parameter as a normal scalar variable. This subroutine applies uppercase formatting to a hash reference's value by accepting two scalars as parameters, the reference and key.

TYPE THIS

```

sub UpperCaseHashRefByKey {
    my ( $ref, $key ) = @_;
    $ref->{ $key } = uc $ref->{ $key };
}
&UpperCaseHashRefByKey( $produceColors,
    "banana" );

```

There is no return statement in this subroutine; it is updating the referenced hash directly with the output of the **uc** function.

RESULTS

If you were to call this subroutine before you deconstructed the hash reference, the output of the program would be:

```

Sorted list of all produce: apple, banana, carrot
Sorted list of all colors: YELLOW, orange, red

```

The function **sort** also responds differently. Alphabetically, it interprets capital letters before lowercase letters. If you were to call this subroutine *after* you deconstructed the hash reference, there would be no uppercase color displayed, but the original hash reference, `$produceColors`, would still be updated.

Nest Variable Types with References

Because all arrays and hashes store their values as scalars, and all array or hash references are themselves stored as scalars, you can nest arrays and hashes together to create *multi-dimensional references*. A nested reference begins like an array or hash reference, depending on what the top-level variable type is. When building a new multi-dimensional reference, it is best to start with a blank slate. A hash-only multi-dimensional reference is the easiest to construct.

```
my $hashref = {}; # Create a blank hash reference
```

From here you can start adding data arbitrarily.

```
$hashref->{ 'key1' } = [ LIST ];
$hashref->{ 'key1' }->[ index ] = { KEY =>
    VALUE, ... };
$hashref->{ 'key2' } = { KEY => VALUE, ... };
```

When nesting references, it is important to pay close attention to what exists where. In this example, `key1` holds a second-dimensional array reference, and `key2` holds a second-dimensional hash reference. It is fairly easy to construct ad hoc data such as this in Perl, but you need to be careful when deconstructing and reading it back.

It is also possible to share references anywhere within a nested reference. by creating an *alias* to another reference point. An alias makes the same data is accessible from multiple points in the nested reference.

```
$hashref->{ 'key2' }->{ 'array' } =
    $hashref->{ 'key1' };
```

This is where the `Data::Dumper` module can be extremely useful. If at any point you are confused about what data type exists where, you can use the `Dumper` function to examine the entire nested reference.

Nest Variable Types with References

- 1 Create a new Perl script in your editor.
- 2 Type `use Data::Dumper;` to import the `Data::Dumper` module.
- 3 Type `my $hashref = {};` to create a new hash reference.
- 4 Type `$hashref->{ KEY } = {};` to begin a new hash reference under a key.
- 5 Assign new keys and values into the new hash reference.
- 6 Type `$hashref->{ key }->{ key } = [];` to create a new array reference under a hash reference.
- 7 Assign new values into the new array reference.
- 8 Type `$hashref->{ key } = {` to begin a new hash reference. Do not close it yet!
- 9 Type `key => [LIST]`, to create a new array reference called 'color', with assigned values.
- 10 Type `key => VALUE`, to create a simple scalar reference, with an assigned value.

```
1 #!/usr/bin/perl
2
3 use Data::Dumper;
4
5 my $produce = {};
6
7 $produce->{ 'apple' } = {};
8 $produce->{ 'apple' }->{ '...'
9 $produce->{ 'apple' }->{ '...'
10
```

```
11 $produce->{ 'apple' }->{ 'varieties' }
12 $produce->{ 'apple' }->{ 'varieties' }
13 $produce->{ 'apple' }->{ 'varieties' }
14 $produce->{ 'apple' }->{ 'varieties' }
15
16 $produce->{ 'watermelon' } = {
17   color => [ 'green', 'red', 'black' ],
18   type => 'fruit',
19 };
20
```

- 11 Type **key => [** to begin a new array reference. Again, do not close it yet!
- 12 Assign new values into the new array reference.
- 13 Type **]**, to close the array reference.
- 14 Type **;** to close the parent hash reference and the statement.
- 15 Type **print Dumper(\$hashref);** to display the full reference content.
- 16 Save the Perl script.
- 17 Open a Command Prompt in the directory where you saved the Perl script.
- 18 Execute the Perl script.

A multi-dimensional array with nested data types is built, and then displayed with Data::Dumper.

- The output of the first simple hash reference.
- The output of the second complex hash reference.

```

1 #!/usr/bin/perl
2
3 use Data::Dumper;
4
5 my $produce = {};
6
7 $produce->{ 'apple' } = {};
8 $produce->{ 'apple' }->{ 'color' } = 'red';
9 $produce->{ 'apple' }->{ 'type' } = 'fruit';
10 $produce->{ 'apple' }->{ 'varieties' } = [];
11 $produce->{ 'apple' }->{ 'varieties' }->[ 0 ] = 'Granny-Smith';
12 $produce->{ 'apple' }->{ 'varieties' }->[ 1 ] = 'Mcintosh';
13 $produce->{ 'apple' }->{ 'varieties' }->[ 2 ] = 'Gala';
14 $produce->{ 'apple' }->{ 'varieties' }->[ 3 ] = 'Fuji';
15
16 $produce->{ 'watermelon' } = {
17     color => [ 'green', 'red', 'black' ],
18     type => 'fruit',
19     varieties => [
20         'Carolina Cross', 'Densuke', 'Yellow Crimson', 'Moon and Stars',
21     ],
22 };
23
24 print Dumper( $produce );
25

```

```

H:\VB\Perl_Apache_VB\chapter8>
$VAR1 = {
    'apple' => {
        color => 'red',
        type => 'fruit',
        varieties => [
            'Granny-Smith',
            'Mcintosh',
            'Gala',
            'Fuji'
        ]
    },
    'watermelon' => {
        color => [
            'green',
            'red',
            'black'
        ],
        type => 'fruit',
        varieties => [
            'Carolina Cross',
            'Densuke',
            'Yellow Crimson',
            'Moon and Stars'
        ]
    }
};
H:\VB\Perl_Apache_VB\chapter8>

```

Apply It

Appending or updating entries to an established nested reference is not necessarily complex. If you are updating an array or hash value, the format used in the 'apple' example will work for 'watermelon' as a separate statement. However, if you are *appending* to an array, then you need to push the data into a dereferenced array reference.

If you want to add a variety to the watermelon 'varieties' array, then you must identify the correct location in the nested reference to dereference. Using the output of the Dumper function, you can see that \$produce->{ 'watermelon' }->{ 'varieties' } needs to be temporarily dereferenced with @{\$sarrayref}.

TYPE THIS

```

push( @{$produce->{ 'watermelon' }->{
    'varieties' } },
    "Cream of Saskatchewan" );

```

RESULTS

Data::Dumper's output now shows that there are five varieties of watermelon, instead of the four that were originally declared.

Introducing Perl Modules

You use Perl modules to add specific or complex code, often written by a third party, to a specialized script, written by you. These modules often provide a generic interface so that you can use them to simplify complex tasks. Perl ships with several modules that are standardized on all Perl distributions.

Perl developers often identify a complex requirement, create a Perl module to simplify the execution, and release the module to the public through the Comprehensive Perl Archive Network (CPAN). For more information on using CPAN, see Chapter 9.

The @INC Array

When importing modules in Perl, a special built-in array called `@INC` contains a list of all the directories that will be searched. The default content of `@INC` varies, depending on the operating system, Perl distribution, and Perl version. Normally, you do not need to worry about the contents of `@INC` on a specific system. When a third-party module is installed, it automatically places itself within the context of `@INC`.

Identify All Perl Modules on a System

Perl ships with several modules that are installed along with the main Perl interpreter. This program examines all files found under `@INC`, and converts the path and filename into the suitable module name. Each outputted line is a valid module that you can reference with the `use` function in any Perl script on the system.

```
use File::Find;
foreach my $dir ( @INC ) {
    find( sub {
        return if ( $dir eq '.' );
        return unless ( s/\.pm$/ / );
        $_ = $File::Find::dir . $_;
        s/^\$dir///;
        s/\/\:\/\/g;
        print "$_\n";
    }, $dir );
}
```

Creating LOCAL Modules

When developing multiple Perl scripts for a project, the observant developer may recognize that the same basic code exists in several files. It makes sense to generalize that code into a common library and reference that library within each individual script. Most modules that are provided by upstream providers, such as CPAN, are installed globally on a system. However, locally developed modules may exist in the same base directory path as the script and be referenced in the same `use` syntax.

Naming a New Module File

The new module's filename and directory path affect how it is imported and referenced in a Perl script. All module files should end with a `.pm` extension, and may exist in the same directory as the actual Perl scripts, or in a series of subfolders.

For example, a module named `MyModule.pm` would be imported into a Perl script as

```
use MyModule;
```

If a module file is found in a subdirectory, directory paths are separated with two colons (`::`). For example, a directory named `Common` could contain module file `MyModule.pm`; this would be imported as

```
use Common::MyModule;
```

Parts of a Module File

The Perl module file always contains the same four basic parts: a `package` statement, a new subroutine, a series of subroutines exported as methods, and a `return` code.

Declare the Module Name

The first line of every module should be a `package` statement. Typically this matches the `use` statement, and is relative to where the module file will be saved.

```
package Common::MyModule;
```

You can immediately follow the `package` statement with more `use` functions (if any dependency modules are required) as well as any global variables.

Create the Module Constructor Subroutine

You must create a special subroutine called `new` which will be used to initialize your module within a Perl script. The script executes this subroutine through the `new` constructor. The constructor is required to generate an instance of the module and store it as a scalar handle.

```
sub new {
    my $class = shift;
    my $self = {};
    bless( $self, $class );
    return $self;
}
```

The Exported Methods

An exported method is just like any regular subroutine, but with one key difference: the first parameter is always the `$self` hash reference.

```
sub MyFunction {
    my $self = shift;
    [...]
}
```

The `$self` variable is actually a conduit for all data that can be shared within the instance of the module and your Perl script. Any exported method can modify any other variable within its own `$self` reference; that information is persistently kept alive for as long as the module is active in a Perl script.

End the Module

The last line in every module should be a `return` code. You use this to signify to the parent Perl script that the module file is good and no errors were found when importing. Simply put, most modules literally include this as their last line: `1;`. The actual function name, `return`, is implied.

Importing Modules

Some modules automatically export functions when imported into your Perl script, others require you to create a module handle and use that to access its methods in an object-orientated fashion. Be aware of how the module works by reading its documentation.

Reading a Module's Documentation

Most third-party modules that are installed globally are shipped with documentation that describes how they run. You can access that documentation using a command such as `perldoc MODULE`.

Importing a Module

The first line is always the `use` function, regardless of whether the module is global or local.

```
use MODULE;
```

Initializing a Module

Some modules automatically export new functions whenever they are imported (for example, `Data::Dumper`), but most require you to assign a scalar as a *handle* to the module. The handle acts as conduit to all functionality contained within the module.

The handle is often created using the module's `new` constructor function. A module may require arguments when it is initialized, these are passed as parameters to `new`.

```
my $h = MODULE->new( ARGS );
```

Sometimes you will find that modules initialize themselves by using `new` as an actual function. Both examples have the same meaning, and are interchangeable.

```
my $h = new MODULE( ARGS );
```

From here, all methods provided by the module are available as functions through the handle. Depending on the method, additional parameters may be supplied through the handle interface, just like any other subroutine or built-in function.

```
$h->method( PARAMS );
```

Always refer to a module's documentation for the correct way to use the module's methods.

Create a New Module

It is useful to create a new module in Perl if an opportunity exists to re-use the same basic code in multiple scripts in your project, or even in multiple projects. This helps you to generalize common functionality and logic, and centralize it, making the same code accessible from multiple Perl scripts.

Every module must begin with a `package` declaration as its first line, this is used in place of the standard shebang (`#!/usr/bin/perl`) header. The filename itself should match the `package` declaration, ending with a `.pm` extension. All modules must have at least one subroutine called `new`. This acts as the starting-point for the module initialization process. When you assign a module a handle, `new` is executed, which generates the handle constructor for the recipient scalar. The important `$self` variable is established within the `new` subroutine. This

allows all other subroutines contained within the module to act as a series of methods, each sharing data with each other through `$self`. All other subroutines in your module need to accept the `$self` variable as the first parameter. This allows for all modules to share data back and forth related to an initialized instance of the module itself.

Finally, all modules must end with a “true statement” as the last line in the file. Usually this is accomplished simply by typing `1;` as the last line. It is possible for modules to allow you to create functions and variables that are automatically exported whenever the module is imported. Sometimes, a module may require specific code that must be executed when the module is imported, or when the program is finished. This can happen through the special `BEGIN` and `END` subroutines. For more information on these advanced module features, run the `Perldoc` command, `perldoc perlmod`.

Create a New Module

- 1 Open a new file in a text editor.
- 2 Type **package Module;** to introduce the module package.

Note: Remember that the module's package name should match its filename, with the only difference being that the filename should end with the `.pm` extension.

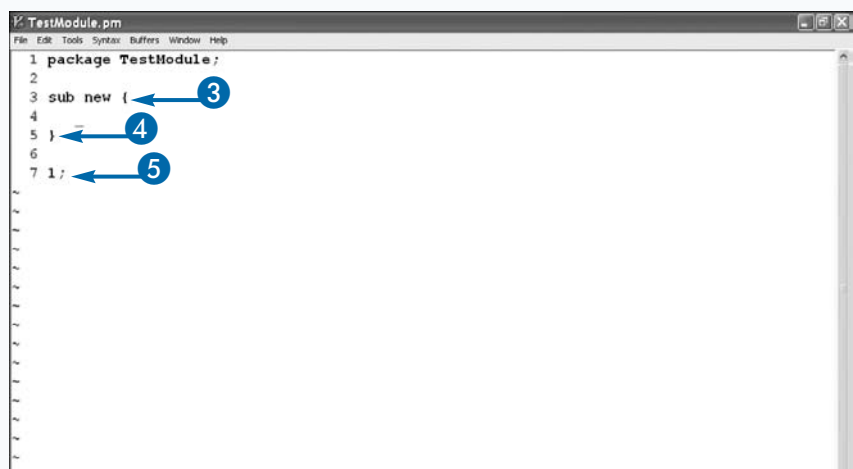
- 3 Type **sub new {** to begin the module initialization subroutine block.
- 4 Type **}** to close the block.

Note: Every module should have exactly one `new` subroutine.

- 5 Type **1;** as the last line in the module.



```
TestModule.pm
File Edit Tools Syntax Buffers Window Help
1 package TestModule;
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



```
TestModule.pm
File Edit Tools Syntax Buffers Window Help
1 package TestModule;
2
3 sub new {
4
5 }
6
7 1;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

- 6 Type **my \$class = shift;** to include the module's class name within a variable.
- 7 Type **my \$self = {};** to create a blank hash reference.
- 8 Type **bless(\$self, \$class);** to bind the blank hash reference to the class.
- 9 Type **return \$self;** to return the blessed hash reference to the parent Perl script.
- 10 Type **sub Function {** to start a new code block for a shared method.
- 11 Type **my \$self = shift;** to import the shared `self` module.
- 12 Type **}** to close the method's code block.
- 13 Save the Perl module as `Module.pm` so it matches its package name.

The new module and its code can now be imported into regular Perl scripts.

```

1 package TestModule;
2
3 sub new {
4     my $class = shift;
5     my $self = {};
6
7     bless( $self, $class );
8
9     return $self;
10 }
11
12 1;

```

```

1 package TestModule;
2
3 sub new {
4     my $class = shift;
5     my $self = {};
6
7     bless( $self, $class );
8
9     return $self;
10 }
11
12 sub Function {
13     my $self = shift;
14
15     print "Running a method from $self\n";
16 }
17
18 1;

```

Extra

Perl modules can be hierarchal in structure, allowing you to define a parent class prior to your module. If you do this, make sure that the module's parent directory name matches the class name. For example, a module named `TestModule` must be stored as `TestModule.pm`, and `Net::TestModule` must be stored under a `Net` directory as `TestModule.pm`. The location of the module file, and class directory, must be somewhere under the `@INC` array. It is this array which Perl uses to search for modules imported by your code.

The `$self` hash reference is persistently available throughout the entire life of a Perl module. It is possible to store *static* data within it, and retrieve it later. The easiest way to do this is to initialize `$self` with content at the module's `new` subroutine with data that could be used later with a module's subroutines.

```
my $self = { @_ };
```

This allows the `new` subroutine to transfer all parameters sent to it into the `$self` hash reference. Now, when the module is initialized, as demonstrated in the next section, any parameters that are passed to `new` are implicitly available in `$self` and all of the module's functions.

Call a Module's Subroutines as Methods

Before a Perl script can take advantage of the subroutines contained within a module, a module reference needs to be established. Just like an array or hash reference, a *module reference* is a scalar that points to a particular instance of a module. That scalar then acts as a handle to the module's contents, including its variables, methods, and shared `$self` hash reference.

Naturally, a single script may have many module instances running in tandem, even multiple instances of the same module, if required. It is the individual scalar that holds the module reference that keeps everything organized.

It is that same scalar that is used to access the module's subroutines as methods. This happens through an arrow

(`->`), similar to what you saw earlier in this chapter when dereferencing a reference.

```
use Module;
my $h = Module->new();
$h->method();
```

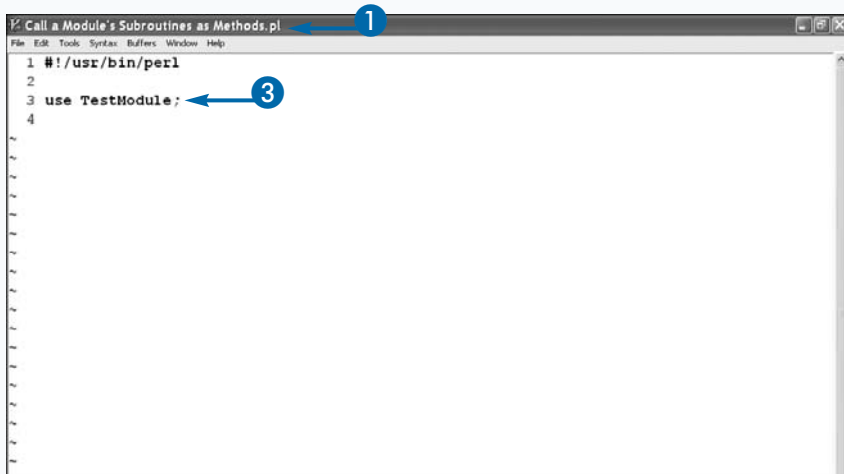
You can even use the handle to access the same `$self` variable used within the actual module. In the exact same way that subroutines are executed as modules, variables can be manipulated using the handle.

```
$h->{ KEY } = VALUE;
```

Regardless of whether your module utilizes `KEY` anywhere, the script can use it to store additional data, just like any type of complex hash reference. When the program ends, all active module references are automatically released from memory. It is possible to manually destroy a module by calling `undef` on the module reference.

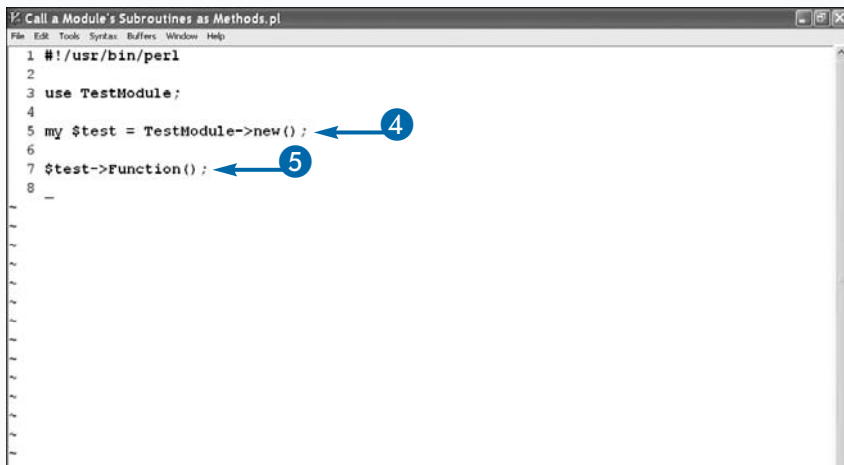
Call a Module's Subroutines as Methods

- 1 Open a new Perl script in a text editor.
- 2 Identify a Perl module that you want to load.
- 3 Type `use Module;`



```
1 #!/usr/bin/perl
2
3 use TestModule;
4
5 ~
6 ~
7 ~
8 ~
9 ~
10 ~
```

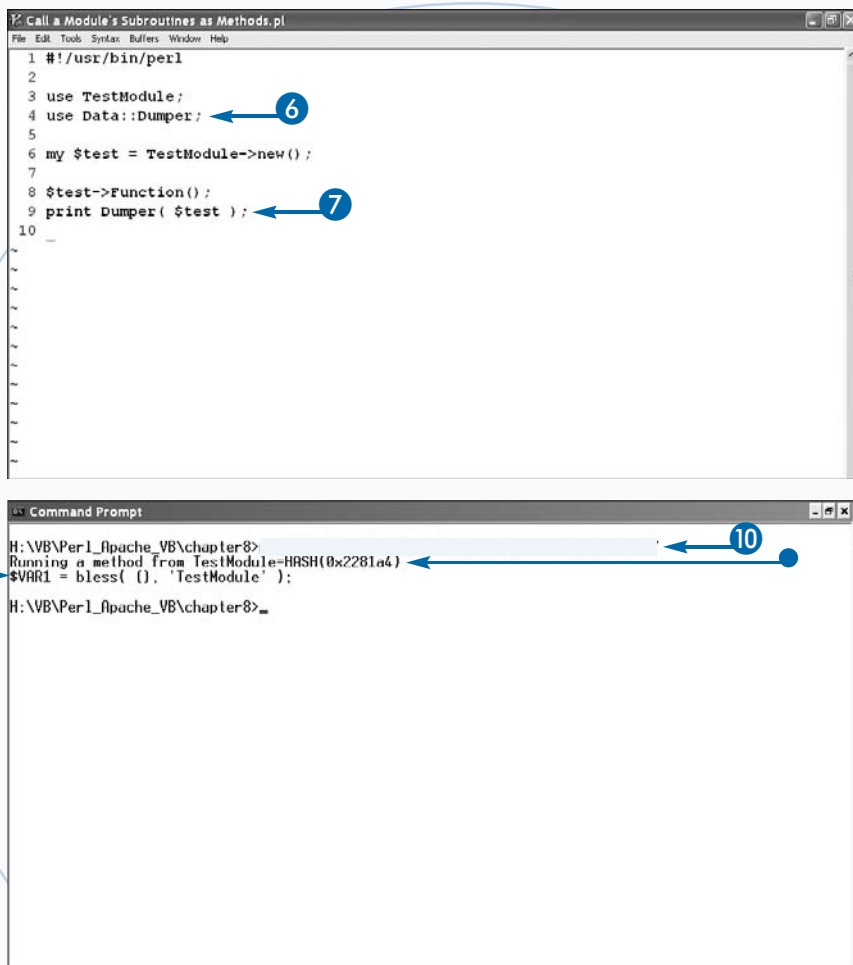
- 4 Type `my $handle = Module->new();` to declare a new scalar, initialize the module, and store the module reference.
- 5 Type `$handle->Function();` to execute one of the exported functions in the module.



```
1 #!/usr/bin/perl
2
3 use TestModule;
4
5 my $test = TestModule->new();
6
7 $test->Function();
8
9 ~
10 ~
```

Note: Even though `Function` is a subroutine, it is not correct to precede it with an ampersand.

- 6 Type `use Data::Dumper;`
- 7 Type `print Dumper ($handle)`; to examine the contents of the module handle.
- 8 Save the Perl script.
- 9 Open a Command Prompt in the same directory with your Perl script.
- 10 Execute the Perl script.
The module is imported, and its subroutine is executed as a method in your Perl script.
 - Output of the Perl module function.
 - Output of the `Dumper`, showing the contents of the module handle.



Extra

It is possible to pass additional variables to the module's functions, just like any other subroutine. The only difference is that the `$self` variable is the first parameter in the `@_` array. If your function allows for other parameters, then you can write them as

```
sub Function {
    my ( $self, $param1, $param2 ) = @_;
    [...]
}
```

When `Function` is executed within the context of a Perl script, the `$self` parameter is implicitly passed to the method.

```
$test->Function( $param1, $param2 );
```

However, if two methods within a module need to execute each other, they should be treated like regular subroutines, not methods. For example, if `Function` has to call the method `Calculate`, then it must specify `$self` explicitly.

```
sub Function {
    my $self = shift;
    &Calculate( $self, ... );
}
```

Introducing CPAN

The Comprehensive Perl Archive Network (CPAN) is the largest repository of third-party Perl modules, scripts, and documentation that are freely distributed on the Internet. The CPAN repository is also available as a Web site that you can access to search for modules, download source code archives, and review documentation. On your computer, the CPAN repository is also accessible as a command-line program, simply called `cpan`, that you can use to download and install modules with a single command.

CPAN Web Site

You can find the primary CPAN Web site at www.cpan.org. Not only does it contain a searchable database of all available Perl modules, but it is also an excellent resource for example Perl code, documentation, mailing lists, and other useful Web sites.

All Perl distributions have the capability to access CPAN, but CPAN may not be appropriate for all distributions. For example, if you are running ActiveState Perl on Windows, you use the ActivePerl Packager program, `ppm`. For Debian- or Ubuntu-based Linux distributions, you use `apt-get`. For Red Hat- or Fedora-based Linux, you use `yum`. Each of these methods is discussed later in this chapter.

If the module you are looking for does not exist in a format that is usable by your respective packaging program, or your packaging program's version of the module is out of date, then you can use CPAN as a fallback to ensure that it is installed.

For a direct link to the CPAN search engine, go to <http://search.cpan.org> and search by category, author, name, or description. Once you find a module, you can view its documentation and read third-party user reviews and comments about the module.

Running CPAN

Because the CPAN program requires Internet access to query the CPAN repository, you need to configure it the first time you run it on a system. The configuration is relatively automatic, but CPAN may ask you a question if it identifies an anomaly on your system. Note CPAN attempts to install modules globally on a system. It usually requires root access on Unix or Linux.

The CPAN Program

You use the CPAN program as an interactive text-based interface to the CPAN repository online. You can start the CPAN program by opening a terminal and typing

```
cpan
```

Once you launch the CPAN program, you can type **help** to bring up a list of available commands, and type **quit** to close the program. Run `perldoc CPAN` for a complete list of available CPAN commands and documentation.

The CPAN Command-Line

It is possible to bypass the interactive CPAN program and use the command-line interface directly to install modules in one command. This is especially useful if you already know the module's name:

```
cpan MODULE
```

Run `perldoc cpan` for a complete list of CPAN command-line arguments.

Using CPAN

Most interaction with CPAN happens using the program interface. Here you can search, download, compile, test, and install modules using basic commands.

Searching for Modules

If you do not know the exact module name, the CPAN program is not the easiest place to browse for modules. Conduct a search at the CPAN Web site, <http://search.cpan.org>, instead.

If you do already know the module name, you can use various commands to access information about the module in the CPAN program. If you do not know the module name, you can perform a simple search query by regular expression.

COMMAND	DESCRIPTION
<code>m /query/</code>	Search for all modules with <i>query</i> in their name.
<code>m MODULE</code>	Download and display the module's author, version, and date.
<code>readme MODULE</code>	Download and display the module's readme document.
<code>perldoc MODULE</code>	Download and display the module's primary documentation.

Because the query method is a regular expression search, it is possible to narrow down the search criteria to match the query. For example, use `m /^Date:$/` to list all modules in the `Date` class.

Installing Modules

Once you have identified a module, you can install the module with the CPAN program using a single command, `install MODULE`.

The installation process implies other commands to download, compile, and validate a module before it is placed globally on a system.

COMMAND	DESCRIPTION
<code>get MODULE</code>	Downloads the module from CPAN
<code>make MODULE</code>	Compiles the module (implies <code>get</code>)
<code>test MODULE</code>	Validates that the module is working (implies <code>make</code>)
<code>install MODULE</code>	Installs the module globally (implies <code>test</code>)

Because each command is chained, simply running `install MODULE` launches the entire process.

Downloading Modules

The module download process automatically identifies the latest module version available on CPAN and any dependency modules that you may also require. You download modules from the closest CPAN mirror based upon the configuration process.

Compiling Modules

Some Perl modules contain source code written in another language, most often C, in order to provide certain functionality, as determined by the module's author. You need to compile this source code into a binary form that is compatible with your system.

CPAN will display a warning message if it cannot find appropriate build or compile commands and libraries; you must install any missing libraries in order to continue.

If it is not possible to install these libraries, then you need to resort to another package manager that provides pre-built binaries for your system, such as `ppm`, `apt-get`, or `yum`, as described later in this chapter.

Testing Modules

Each CPAN module has a built-in test script that is designed to validate the module to ensure that it is working correctly. The tests are largely automatic but may catch or identify bugs related to a module that the author never anticipated.

If a test fails, it is possible to force a module installation using the command, `notest install MODULE`. However, this may install a module on your system that does not work as documented.

Upgrading Modules

One of the original goals of CPAN was to provide an upgrade path for Perl modules installed on systems that lacked a native packaging front-end such as `ppm`, `apt`, or `yum`.

COMMAND	DESCRIPTION
<code>r MODULE</code>	Checks if an update is available on the CPAN repository
<code>upgrade MODULE</code>	Upgrades the module

If you do not supply a module name to either of these commands, CPAN checks for all modules that are upgradeable, and then performs the upgrade.

Uninstalling Modules

Unfortunately, CPAN does not contain a standard way to actually uninstall a module. Generally speaking, searching for the module's source code in the path's referred by the `@INC` array and deleting it does work, but this is not exactly ideal.

Instead, an upgraded CPAN program is available to provide this functionality, CPANPLUS. You can execute this program from the terminal, just like the CPAN program, but with the command

```
cpanp
```

The CPANPLUS program provides the same functionality as the CPAN program, with the added benefit of the `uninstall` command, `u MODULE`.

You can find more information about CPANPLUS, and the complete list of available functionality, by using the `Perldoc` command, `perldoc cpanp`.

Configure CPAN

By default, when you first execute a CPAN program, it attempts to configure itself for the local system. This process is largely automatic; it identifies an appropriate CPAN mirror, any available support programs, and other options.

Most CPAN configuration defaults are acceptable to most users, but if it makes a mistake, or if you want to select an alternative option, it is useful to configure CPAN to act accordingly. Because the CPAN program requires Internet access, it is important to select the correct options dealing with protocol, network proxy, and access rights.

The automated CPAN configuration process should be sufficient for most users, but may not provide the intended results on some systems. You may need to perform some supplemental configuration beyond the automatic configuration program.

Configure CPAN

- 1 Type **cpan** to launch the CPAN program.

Note: The CPAN program command is standard across Windows and Linux platforms.

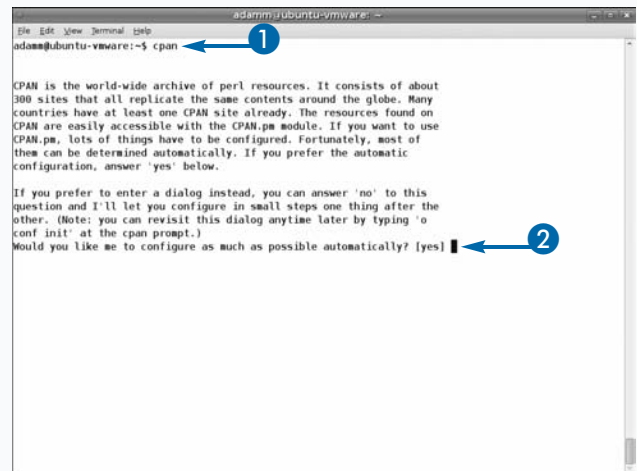
- 2 Press Enter to configure CPAN automatically.

CPAN automatically configures itself.

- 3 Type **exit** to quit the CPAN program.

Extra

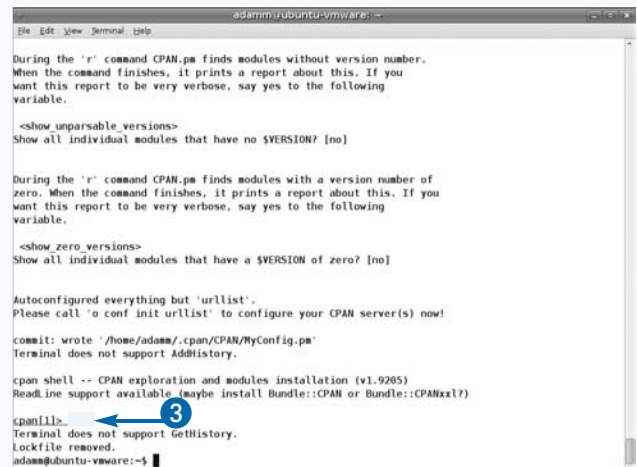
Type **o conf** to open a full list of available configuration options, or type **o conf help** for a basic help message. You can run the command **o conf init urllist** to select a CPAN mirror. Linux-based systems require root access in order to install Perl modules globally. You need to instruct CPAN to use root if you run the program as a non-privileged user. To do this, in the CPAN program run the command, **o conf init make_install_make_command**. If you know the system's root password, set this option to **su root -c make**; otherwise, use **sudo make**. Finally, if you manually changed any configuration options, run **o conf commit** to save the changes for future CPAN sessions.



```
adam@ubuntu-vmware:~$ cpan

CPAN is the world-wide archive of perl resources. It consists of about
300 sites that all replicate the same contents around the globe. Many
countries have at least one CPAN site already. The resources found on
CPAN are easily accessible with the CPAN.pm module. If you want to use
CPAN.pm, lots of things have to be configured. Fortunately, most of
them can be determined automatically. If you prefer the automatic
configuration, answer 'yes' below.

If you prefer to enter a dialog instead, you can answer 'no' to this
question and I'll let you configure in small steps one thing after the
other. (Note: you can revisit this dialog anytime later by typing 'o
conf init' at the cpan prompt.)
Would you like me to configure as much as possible automatically? [yes]
```



```
During the 'r' command CPAN.pm finds modules without version number.
When the command finishes, it prints a report about this. If you
want this report to be very verbose, say yes to the following
variable.

<show_unparsable_versions>
Show all individual modules that have no $VERSION? [no]

During the 'r' command CPAN.pm finds modules with a version number of
zero. When the command finishes, it prints a report about this. If you
want this report to be very verbose, say yes to the following
variable.

<show_zero_versions>
Show all individual modules that have a $VERSION of zero? [no]

Autoconfigured everything but 'urllist'.
Please call 'o conf init urllist' to configure your CPAN server(s) now!

commit: wrote '/home/adam/.cpan/CPAN/MyConfig.pm'
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.9205)
Readline support available (maybe install Bundle::CPAN or Bundle::CPANxxl?)

cpan[]>
Terminal does not support GetHistory.
Lockfile removed.
adam@ubuntu-vmware:~$
```

Search for Perl Modules with CPAN

It is possible to search for Perl modules with the CPAN program. Searching for a module is certainly easier if you happen to know at least part of its name. For example, all date-related modules have “date” in their name. So, the command `m /^Date::/` is a good starting point to list available modules in the “Date” class.

Once you have identified a module’s complete name, you can view the module’s manual page with `perldoc MODULE`, or view the module’s readme document with `readme MODULE`, all without actually installing the module.

Along with using the CPAN program to find modules, the CPAN Web site, <http://search.cpan.org>, is a great resource if you do not know if

a module exists, or its proper name, or even just to randomly browse for what is available.

Extra

Because the CPAN program is an interface to the CPAN repository, you require Internet access for most commands. If you have not used the CPAN program in a while, or are using it for the first time, CPAN downloads the repository’s master index during the first search attempt. This download should happen automatically, but you can force it to refresh with the command `reload index`.

Search for Perl Modules with CPAN

- 1 Open the CPAN program.
- 2 Type `m /query/` to search for a module’s name.

Note: The example query `/^Date::/` will search for all modules that are members of the Date class, or, the modules whose names begin with “Date::”.

The CPAN command runs.

Note: If CPAN has not been configured yet, it asks you a series of questions the first time you use it.

- 3 Identify a module to install.
 - 4 Type `m module` to view the module’s summary.
- CPAN outputs the identified module.

```
adam@ubuntu-vmware:~$ cpan
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.9205)
ReadLine support available (maybe install Bundle::CPAN or Bundle::CPANx1?)

span[1]:
CPAN: Storable loaded ok (v2.18)
Going to read /home/adam/.cpan/Metadata
Database was generated on Mon, 19 Oct 2009 05:27:05 GMT
Module Date::Baha::i (GENE/Date-Baha-1-0.1802.tar.gz)
Module Date::Biorhythm (BEPPU/Date-Biorhythm-2.2.tar.gz)
Module Date::Business (DESIGNER/Date-Business-1.2.tar.gz)
Module Date::Calc (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Calc::Iterator (BRONTO/Date-Calc-Iterator-1.00.tar.gz)
Module Date::Calc::Object (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Calc::PP (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Calc::XS (STBEY/Date-Calc-XS-6.2.tar.gz)
Module Date::Calendar (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Calendar::Profiles (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Calendar::Year (STBEY/Date-Calc-6.2.tar.gz)
Module Date::Chinese (RBOM/Date-Chinese-1.03.tar.gz)
Module Date::Christmas (HFB/Date-Christmas-1.02.tar.gz)
Module Date::Components (DMAC/Date-Components-0.2.1.tar.gz)
Module Date::Convert (MORTY/Date-Convert-0.16.tar.gz)
Module Date::Convert::French_Rev (JFDRGET/Date-Convert-French_Rev-0.05.tar.gz)
Module Date::Converter (ANDY/Date-Converter-1.1.tar.gz)
Module Date::Converter::Alexandrian (ANDY/Date-Converter-1.1.tar.gz)
Module Date::Converter::Armenian (ANDY/Date-Converter-1.1.tar.gz)
Module Date::Converter::Bahai (ANDY/Date-Converter-1.1.tar.gz)
Module Date::Converter::Coptic (ANDY/Date-Converter-1.1.tar.gz)
Module Date::Converter::Ethiopian (ANDY/Date-Converter-1.1.tar.gz)
```

```
adam@ubuntu-vmware:~$ cpan
Module Date::Simple::IDB (IZUT/Date-Simple-3.03.tar.gz)
Module Date::Simple::Fat (IZUT/Date-Simple-3.03.tar.gz)
Module Date::Simple::ISO (IZUT/Date-Simple-3.03.tar.gz)
Module Date::Simple::Month (HORIUCHI/Date-Simple-Month-0.02.tar.gz)
Module Date::Simple::Range (ZUMMO/Date-Simple-Range-1.1.tar.gz)
Module Date::Span (RJBS/Date-Span-1.125.tar.gz)
Module Date::Spoken::German (CHREIN/Date-spoken-german/Date-Spoken-German-0.05.tar.gz)
Module Date::SundayLetter (RBOM/Date-SundayLetter-1.10.tar.gz)
Module Date::Tie (FGLDCK/Date-Tie-0.20.tar.gz)
Module Date::Time (N/A)
Module Date::Tiny (ADANK/Date-Tiny-1.04.tar.gz)
Module Date::Tolkien::Shire (TBRAUN/Date-Tolkien-Shire-1.13.tar.gz)
Module Date::Transform (CTBROWN/Date-Transform-0.11.tar.gz)
Module Date::Transform::Closures (CTBROWN/Date-Transform-0.11.tar.gz)
Module Date::Transform::Constants (CTBROWN/Date-Transform-0.11.tar.gz)
Module Date::Transform::Extensions (CTBROWN/Date-Transform-0.11.tar.gz)
Module Date::Transform::Functions (CTBROWN/Date-Transform-0.11.tar.gz)
Module Date::WeekOfYear (GNG/Date-WeekOfYear-1.04.tar.gz)
204 items found

span[2]:
span[3]:
```

Install Perl Modules with CPAN

You can use CPAN to install Perl modules, making it possible to download and install them with a single command. Sometimes, a Perl module may require a specific *build* and *install* process. CPAN simplifies this process by combining any preparation steps into one command, `install MODULE`.

When you use the `install` command, the CPAN program automatically executes other commands, prior to installing a package. This includes `get` to download the package, `make` to build it, and `test` to run the module's test suite. As a result, the `install` command implies `test`, which implies `make`, which implies `get`, so all you need to remember is `install`.

You can run the `install` command either within the CPAN program, or directly from the command-line. However, you do not need the actual `install` keyword; just run `cpan MODULE` in

a DOS Prompt or Terminal window. The install process may require elevated privileges if you are going to install the modules globally on the system, which is the default functionality. This is usually the case on Unix systems. Using CPAN on Windows does not usually require special privileges.

Remember, if you are using a Debian-, Ubuntu-, or Red Hat-based system, use the module installation method designed for your particular situation. You should only install through CPAN if a DEB or RPM package does not exist, or is out of date, or if you are running on a system that does not support these types of packages.

If you choose to install a CPAN package on an ActiveState Perl installation, then ActivePerl Package Manager properly recognizes CPAN-installed modules so that they do not conflict with PPM-installed modules. However, PPM-installed modules usually have priority.

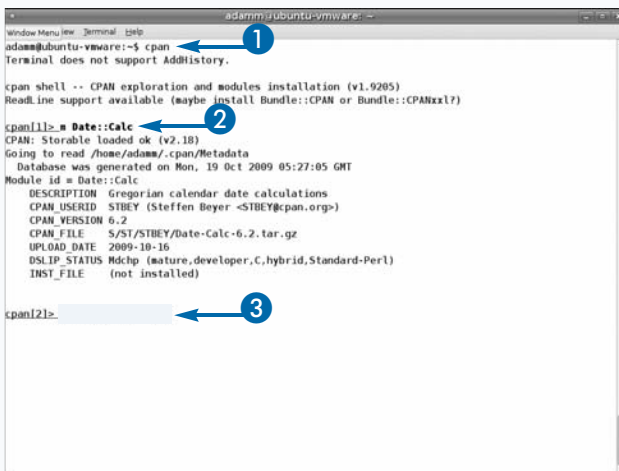
Install Perl Modules with CPAN

- 1 Run the CPAN program.
- 2 Identify a module to install by searching for it.

Note: For more information, see the section, “Search for Perl Modules with CPAN.”

- 3 Type `install module`.

The module install process begins.

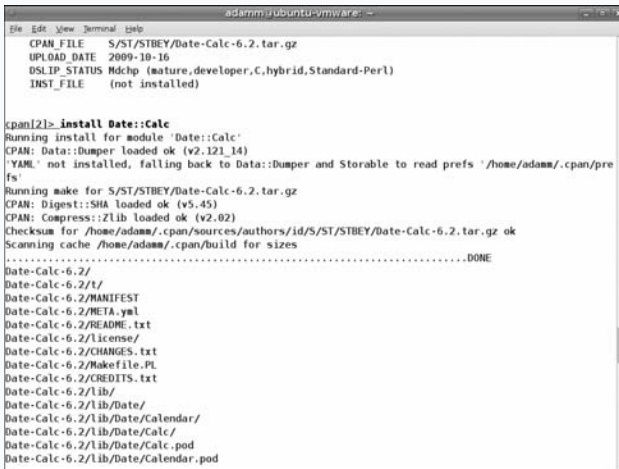


```
adammm@ubuntu-vmware: ~
Window Menu View Terminal Help
adammm@ubuntu-vmware:~$ cpan
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.9205)
Readline support available (maybe install Bundle::CPAN or Bundle::CPANxx1?)

cpan[1]> s Date::Calc
CPAN: Storable loaded ok (v2.18)
Going to read /home/adamm/.cpan/Metadata
Database was generated on Mon, 19 Oct 2009 05:27:05 GMT
Module id = Date::Calc
DESCRIPTION Gregorian calendar date calculations
CPAN_USERID STBEY (Steffen Beyer <STBEY@cpan.org>)
CPAN_VERSION 6.2
CPAN_FILE S/ST/STBEY/Date-Calc-6.2.tar.gz
UPLOAD_DATE 2009-10-16
OSLIP_STATUS Rdchp (mature,developer,C,hybrid,Standard-Perl)
INST_FILE (not installed)

cpan[2]>
```



```
adammm@ubuntu-vmware: ~
File Edit View Terminal Help
CPAN FILE S/ST/STBEY/Date-Calc-6.2.tar.gz
UPLOAD_DATE 2009-10-16
OSLIP_STATUS Rdchp (mature,developer,C,hybrid,Standard-Perl)
INST_FILE (not installed)

cpan[2]> install Date::Calc
Running install for module 'Date::Calc'
CPAN: Data:Dumper loaded ok (v2.121_14)
'YAML' not installed, falling back to Data:Dumper and Storable to read prefs '/home/adamm/.cpan/prefs'
Running make for S/ST/STBEY/Date-Calc-6.2.tar.gz
CPAN: Digest::SHA loaded ok (v5.45)
CPAN: Compress::Zlib loaded ok (v2.02)
Checksum for /home/adamm/.cpan/sources/authors/id/S/ST/STBEY/Date-Calc-6.2.tar.gz ok
Scanning cache /home/adamm/.cpan/build for sizes
.....DONE
Date-Calc-6.2/
Date-Calc-6.2/1/
Date-Calc-6.2/MANIFEST
Date-Calc-6.2/META.yml
Date-Calc-6.2/README.txt
Date-Calc-6.2/License/
Date-Calc-6.2/CHANGES.txt
Date-Calc-6.2/Makefile.PL
Date-Calc-6.2/CREDITS.txt
Date-Calc-6.2/lib/
Date-Calc-6.2/lib/Date/
Date-Calc-6.2/lib/Date/Calendar/
Date-Calc-6.2/lib/Date/Calc/
Date-Calc-6.2/lib/Date/Calc.pod
Date-Calc-6.2/lib/Date/Calendar.pod
```


Introducing ActivePerl Perl Package Manager



ActivePerl *Perl Package Manager*, or PPM, is a utility produced by ActiveState to download, install, upgrade, and remove Perl modules, just like CPAN. The primary difference between it and the CPAN repository is that PPM accesses a separate ActiveState repository of pre-built modules as binary packages. The CPAN program attempts to compile packages directly on your computer during the package download-and-install process. However, the ActiveState repository is sourced from the CPAN repository, so the same basic catalog should be available on both.

The main disadvantage of the CPAN program on a Windows system is that you require a C compiler and

libraries to build complex Perl modules from source code. This is not always available, or practical, in a Windows environment, which is why pre-built module packages are preferred.

To PPM's credit, it readily installs the `dmake` program and the MinGW C compiler, as well as support libraries, onto a Windows system if a user ever runs `cpan` on the command-line. This enables you to use CPAN functionality on an ActivePerl system. If CPAN were to contain a more recent version of a module than PPM's version, the option is now available to the user to use CPAN for the upgrade; PPM honors the more recent CPAN version.

Running Perl Package Manager

You can access PPM in two ways: from the command-line or as a Windows program. The choice is a matter of preference, but sometimes you may be on a system where a graphical interface is not available.

Command-Line Interface

You can access the command-line interface for PPM by opening a Terminal window or DOS Prompt and typing in the PPM binary with a command:

```
ppm command
```

You can find the full list of available commands by running

```
ppm help
```

Graphical User Interface

You can access the graphical interface for PPM either through the ActivePerl program group under the Start Menu on Windows, or by running `ppm` without any command-line arguments.

Documentation

You can access the PPM documentation in two forms, man-page and HTML.

Command-Line Interface Manual

The manual to the PPM CLI (Command-Line Interface) program is available from the PerlDoc program:

```
perldoc ppm
```

This brings up the complete list of available command-line arguments to instantly install, upgrade, or remove packages.

Graphical User Interface Manual

You can access the manual to the PPM GUI (Graphical User Interface) program by clicking Help → Contents, or by pressing F1. This opens a Web browser to a local HTML page describing the PPM program, usage, and additional repositories.

Using Perl Package Manager

The process of installing modules is slightly different, depending on the interface method that you use. When using the graphical interface, you first search for a package, then queue it for installation, upgrade, or removal, and finally process the queue.

When using the command-line interface, there is no queue. PPM immediately processes requests to install, upgrade, or remove packages.

Searching for Modules

You actually search for Perl modules by searching for the respective PPM package. Perl modules already have a specific naming style, but the PPM package-naming convention needs a unique model to represent the packaged binary form of a module.

In essence, when you search in PPM you are actually filtering by a query. PPM tests the query against each package in the repository by comparing it to the package name, abstract, or author.

Adding Repositories

It is possible to add additional repositories into PPM beyond the default ActiveState repository. Once you add in a repository, any packages that the repository has available will appear on the main list.

Installing and Removing Packages

In the graphical interface, PPM has a running queue of requested packages to install, upgrade, or remove. PPM flags packages in the interface; then, once you are happy with the request, it applies the list of changes all at once.

PPM is smart enough to not allow an impossible situation by honoring package dependencies. If you request to remove a package that another package depends on, you should remove either both or none at all.

Package Areas

When you install a package, it is placed in a specific *area*. On a fresh installation there are two areas, perl and site. The perl area indicates a package that shipped with Perl. The site area indicates that the package was installed specifically *at this site*, or on this computer.

It is possible to add custom areas into ActivePerl. This allows a user who does not have full rights to the `C:\Perl\site\lib` or `C:\Perl\lib` paths the opportunity to install a package through PPM. The difference is that the new area is not a standard location, as in perl or site, but rather a path under the user's home directory.

Upgrading Packages

PPM automatically identifies any packages that can potentially be upgraded by comparing all installed and available version numbers. PPM can list all available package upgrades on the command-line when you run the command

```
ppm upgrade
```

Using the graphical interface, click View → Upgradeable Packages.

Applying the Queue

Applying the queue is a process that only happens in the graphical interface. Once initiated, PPM processes all requested changes one after another. It downloads all packages that you are installing or upgrading, and deletes all packages that you are removing.

Downloading Binary Packages

You download all binary packages from a repository at the ActiveState servers. The repository is based upon the contents of CPAN, along with their own custom packages, and other non-Perl binary programs.

This is convenient because PPM should not be restricted to only Perl modules. You can use PPM to deliver useful utilities that are not normally available for a particular environment. For example, PPM can deliver and install `gcc`, The GNU C Compiler, and allow you to make executable binaries for Windows.

Configure ActivePerl PPM

It is possible to configure ActivePerl PPM to support additional functionality and to customize your view of the packages installed on your system. You can also customize the location where you install packages, and what repository servers you use.

The easiest change you can apply to ActivePerl PPM is to modify the view of the packages on the repository and your computer. This allows you to compare which packages are installed, which packages you can upgrade, and which packages are available on the repository.

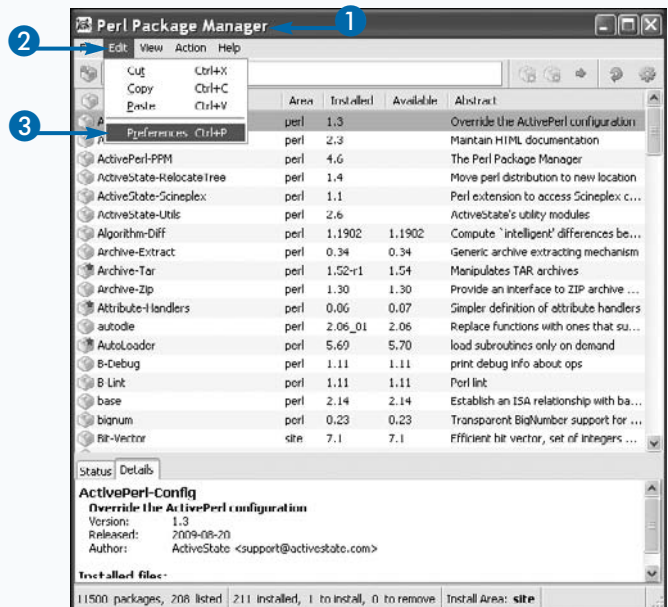
ActivePerl PPM also supports a concept called *install areas*. By default, there are two install areas configured, “site” and “perl”. Modules that ship with the core Perl distribution are pre-installed into the “perl” area. Modules that you install through a third-party program, such as CPAN or ActivePerl PPM, use the “site” area.

Configure ActivePerl PPM

- 1 Open Perl Package Manager.

Note: You can open PPM on Windows by clicking *Start* → *Programs* → *ActivePerl* → *Perl Package Manager*.

- 2 Click Edit.
- 3 Click Preferences.

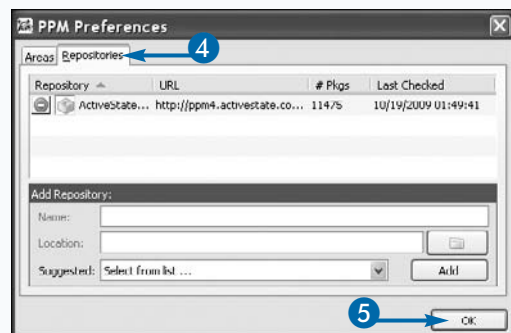



The PPM Preferences dialog box appears, displaying the Areas tab.

- 4 Click the Repositories tab.

The PPM Preferences, Repositories tab appears.

- 5 Click OK.

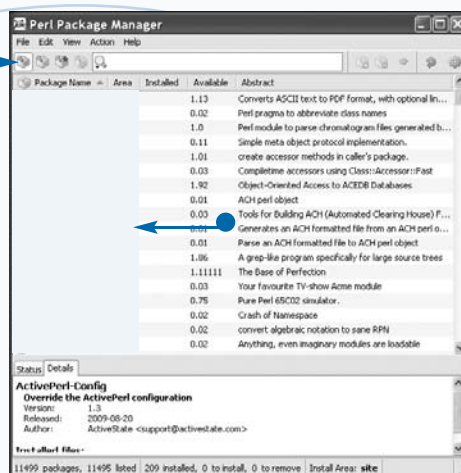



- 6 Click Showallpackages () to show all packages.

PPM displays all available packages.

- Available packages.

Note: Installed packages have an “Installed” version number.




- 7 Click Showinstalledpackages () to show installed packages.

PPM displays all installed packages.

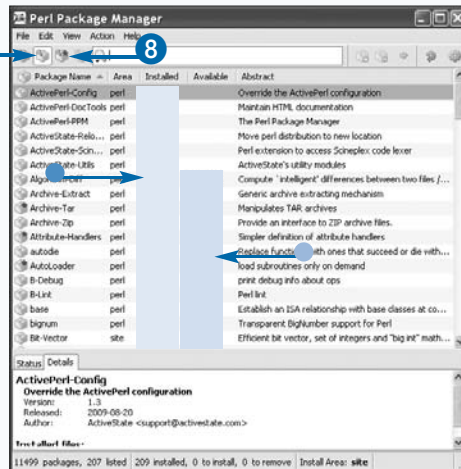
- Installed versions of packages.

Note: Packages with a blank “Available” version have no version on the repository.

- 8 Click Showupgradablepackages () to show upgradable packages.

PPM displays all upgradeable packages.

- Upgradeable versions of packages.



Extra

It is also possible to configure which columns PPM displays in the package list by clicking View → View Columns. Available columns include Area configured, Installed version, Repository, Available version, Abstract description, and Author name.

When adding a new repository, PPM offers a suggested list of potential third-party repositories to use. Adding one of them in is a matter of selecting a “Suggested” repository in the Repositories tab of the PPM Preferences. Once enabled, you can view the packages from that repository by enabling the Repository view column, and then sort the list by clicking the column name.

PPM offers various shortcuts for performing the same configuration tasks.

PPM SHORTCUT	EFFECT
Ctrl+1	View all packages
Ctrl+2	View installed packages
Ctrl+3	View upgradeable packages
Ctrl+P	Open Preferences

Search for Perl Modules with ActivePerl PPM

It is possible to search for Perl modules in ActivePerl PPM by filtering on the package's name or a keyword in the abstract. You can use this feature to quickly locate a package if you only know part of its name or description.

If you want to selectively search by name, abstract, or author, click Searchby. By default, the program searches by the module's name and abstract text. Changing your search terms allows you to narrow down your search.

When Perl modules are packaged up in PPM, their package names follow a standardized naming convention that differs from the original module name. For example, a Perl module normally identified as `Class::Package` would be packaged in PPM as `Class-Package`. In other words, all Perl module packages are described on PPM by replacing `::` with `-`.

Search for Perl Modules with ActivePerl PPM

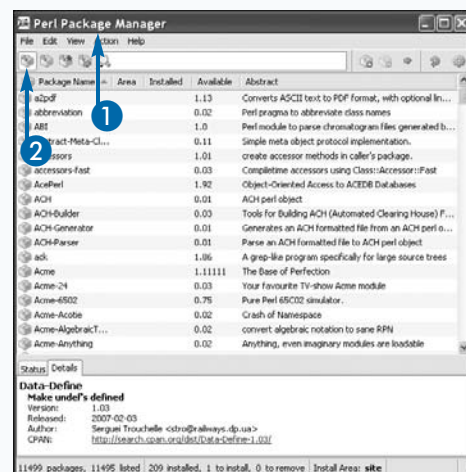
Extra

When using the filter text area, the default search method is by the package's name or a keyword in the abstract. Click Searchby (🔍) to selectively search only by package name, abstract, or author.

It is possible to sort the package list by clicking the column header.

You can use various shortcuts in PPM to perform the same search tasks. Press Ctrl+F to highlight the search window. Press F5 to force PPM to synchronize its database with the installed Perl modules.

- 1 Open Perl Package Manager.
- 2 Click 🔍 to show all available packages.

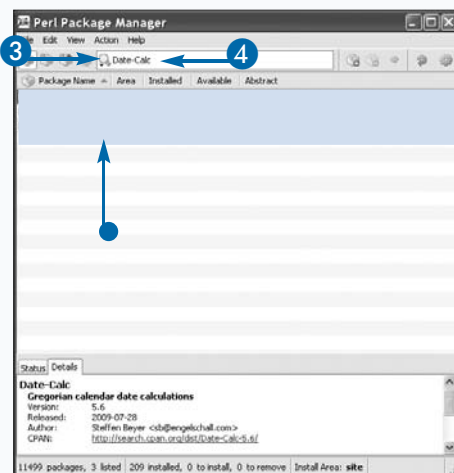


PPM displays all packages.

- 3 Click the text area beside Searchby (🔍).
- 4 Type a term to filter by and press Enter.

PPM applies the search filter query.

- The filtered list of packages.



Install Perl Modules with ActivePerl PPM


In order to install Perl modules through ActivePerl PPM, you must first locate the module's package, mark it for install, and then instruct PPM to install it. You can use this technique to actually install or upgrade a Perl module using ActivePerl PPM.

Once you have queued your packages and selected "run marked action," the system prompts you for confirmation and then processes your request. The installation process involves downloading the PPM package, as well as any dependencies, on to your computer. You need to be connected to the Internet for this to happen.

Once complete, all new modules are available to every Perl script. Simply import the module you want using its standard name.

Install Perl Modules with ActivePerl PPM

- 1 Identify a package to install or upgrade in PPM.
- 2 Right-click the package.
- 3 Click Install *Package version*.

- 4 Click Execute ().

A dialog box appears, asking you to confirm the installation.

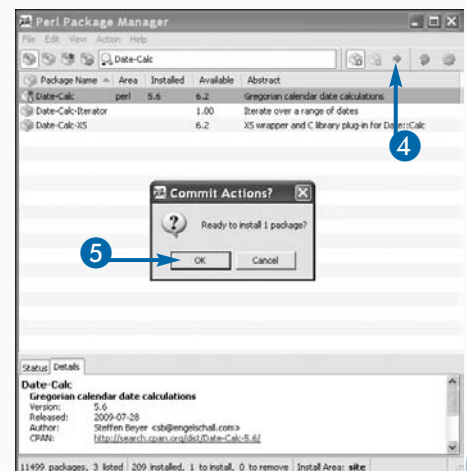
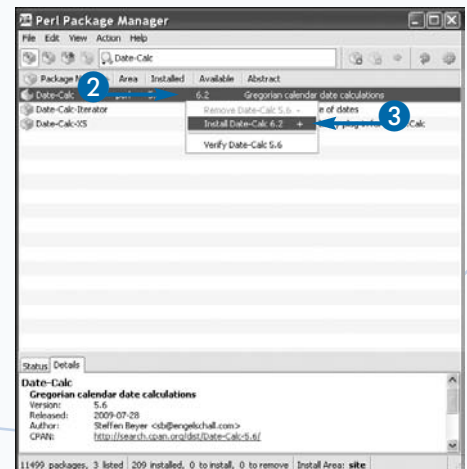
- 5 Click OK.

PPM installs the new package.

Removing packages through PPM uses the exact same method, except that you must explicitly mark the package for removal.

Extra

Installing a package with PPM requires a connection to the Internet to download the index of available packages, and the actual package. If you select a package that requires dependencies, PPM automatically adds all related packages to the install queue. You can use various shortcuts in PPM to perform the same installation tasks. When highlighting an individual package, press + or - to queue the package for installation or removal, respectively. Press Ctrl+Enter to process the queue.



Search for Perl Modules in Debian/Ubuntu Linux

You can search for Perl modules in the Debian- and Ubuntu-based systems by using the APT program to find the module's DEB package. You can use this technique to quickly locate a package if you only know part of its name or description. When Perl modules are converted into a DEB package, their package names follow a standardized naming convention that differs from the original module name. For example, a Perl module normally identified as `Class::Package` would be packaged in Debian as `libclass-package-perl`. In other words, all Perl module packages on Debian/Ubuntu are in lowercase, begin with "lib", end in "-perl", and replace "::" with "-". This section demonstrates the `apt-cache` program as an example, but any APT-compatible search program will do. The module's naming convention remains the same, but the search syntax may be different.

Search for Perl Modules in Debian/Ubuntu Linux

- 1 Type `apt-cache search` `^lib perl$`.

Note: Append `| less` to the command to page through the list.

APT displays the list of available Perl modules.

- 2 Type `apt-cache show` `package`.

- The module's package name.
- The module's version.
- The module's description.

Apply It

The `apt-cache search` command is regular-expression aware. If you add in another keyword without a caret or dollar sign, it is included with your search as "contains the word xyz", which narrows down your search.

TYPE THIS

```
apt-cache search ^lib date perl$
```



RESULTS

APT displays all Perl module packages related to calendar dates.

```
adam@ubuntu-vmware:~$ apt-cache search ^lib perl$
libapache2-reload-perl - Reload Perl modules when changed on disk
libapparmor-perl - AppArmor library Perl bindings
libappconfig-perl - Perl module for configuration file and command line handling
libapt-pkg-perl - Perl interface to libapt-pkg
libarchive-zip-perl - Module for manipulation of ZIP archives
libauthen-sasl-perl - Authen::SASL - SASL Authentication framework
libberkeleydb-perl - use Berkeley DB 4 databases from Perl
libbit-vector-perl - Perl module for bit vectors and more
libbsd-resource-perl - BSD process resource limit and priority functions
libcairo-perl - Perl interface to the Cairo graphics library
libcarp-clan-perl - Perl enhancement to Carp error logging facilities
libchart-perl - Chart Library for Perl
libclass-accessor-perl - Automated accessor generator
libclone-perl - recursively copy Perl datatypes
libcompress-bzip2-perl - Perl interface to Bzip2 compression library
libcompress-raw-zlib-perl - low-level interface to zlib compression library
libconfig-inifiles-perl - Read .ini-style configuration files
libconvert-asn1-perl - Access to the ASN.1 data structures
libconvert-binx-perl - Perl module for extracting data from macintosh Binhex files
libconvert-tnef-perl - Perl module to read TNEF files
libconvert-uulib-perl - Perl interface to the uulib library (a.k.a. uuencode/uuencode)
libcrypt-blowfish-perl - Blowfish cryptography for Perl
libcrypt-openssl-bignum-perl - Access OpenSSL multiprecision integer arithmetic libraries
libcrypt-openssl-random-perl - Access to the OpenSSL pseudo-random number generator
libcrypt-openssl-rsa-perl - Perl module providing basic RSA functionality
libdate-manip-perl - a perl library for manipulating dates
libdbd-mysql-perl - A Perl5 database interface to the MySQL database
libdbd-sqlite3-perl - Perl DBI driver with a self-contained RDBMS
libdbi-perl - Perl Database Interface (DBI)
libdevel-leak-perl - Utility for looking for perl objects that are not reclaimed
libdevel-sydump-perl - Perl module for inspecting perl's symbol table
```

```
adam@ubuntu-vmware:~$ apt-cache show libdate-manip-perl
Package: libdate-manip-perl
Priority: optional
Section: perl
Installed-Size: 492
Maintainer: Ubuntu Core Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Debian Perl Group <pkg-perl-maintainers@lists.alioth.debian.org>
Architecture: all
Version: 5.54-1
Depends: perl (>= 5.6.0-16)
Filename: pool/main/libd/libdate-manip-perl/libdate-manip-perl_5.54-1_all.deb
Size: 172984
MD5sum: 1f817d22eb98b8e14a4a05b1404888
SHA1: 884fd26727919946c33ad84a22702a27cd3add9
SHA256: f4fc289afa67b18187b88ae116a88464de332705800bfe519233a0b71d39b52

Homepage: http://search.cpan.org/dist/Date-Manip/
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Origin: Ubuntu
Task: mythbuntu-backend-master, mythbuntu-backend-slave, mythbuntu-desktop, mythbuntu-frontend

adam@ubuntu-vmware:~$
```

Install Perl Modules in Debian/Ubuntu Linux

It may seem redundant to search and install Perl modules as a Debian package rather than through CPAN; however, this pre-packaged method is preferred. The Debian maintainers strive to ensure every program works as designed on a Debian system as they do anywhere else. If there is a quirk on Debian that warrants a source-code change in the module, but the change only applies to Debian-based systems, the DEB package will have the change but CPAN will not.

You can also easily upgrade and remove Debian packages. Debian packages also have inherent dependency checking. If a module depends on another third-party module or program, the DEB module installation process properly downloads and installs any dependencies.

Apply It

If after installing a module you are not sure how to use it in a Perl script, you can use the `perldoc` program to open the module's manual. However, `perldoc` is not aware of DEB package names, only the names of Perl modules.

TYPE THIS

```
perldoc Date::Manip
```

RESULTS

The `Date::Manip` module's documentation displays.

Install Perl Modules in Debian/Ubuntu Linux

1 Type **`sudo apt-get install package`**.

2 Press **y** to confirm the download and installation, if asked.

- APT downloads the Perl module package.
- APT installs the Perl module package.

```
adam@ubuntu-vmware:~$ sudo apt-get install libdate-manip-perl
[sudo] password for adam:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libdate-manip-perl
0 upgraded, 1 newly installed, 0 to remove and 710 not upgraded.
Need to get 173kB of archives.
After this operation, 504kB of additional disk space will be used.
Do you want to continue [Y/n]? 
```

```
adam@ubuntu-vmware:~$ sudo apt-get install libdate-manip-perl
[sudo] password for adam:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libdate-manip-perl
0 upgraded, 1 newly installed, 0 to remove and 710 not upgraded.
Need to get 173kB of archives.
After this operation, 504kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://mirrors.kernel.org karmic/main libdate-manip-perl 5.54-1 [173kB]
Fetched 173kB in 0s (233kB/s)
Selecting previously deselected package libdate-manip-perl.
(Reading database ... 123301 files and directories currently installed.)
Unpacking libdate-manip-perl (from .../libdate-manip-perl_5.54-1_all.deb) ...
Processing triggers for man-db ...
Setting up libdate-manip-perl (5.54-1) ...
adam@ubuntu-vmware:~$
```


Search for Perl Modules in Red Hat Linux

It is possible to search for Perl modules in Red Hat-based systems by using the YUM program to find the module's RPM package. You can use this method to quickly locate a package if you only know part of its name or description.

When Perl modules are converted into an RPM package, their package names follow a standardized naming convention that differs from the original module name. For example, a Perl module normally identified as `Class::Package` would be packaged in Red Hat as `perl-ClassPackage` or `perl-Class-Package`. In other words, all Perl module packages on Red Hat begin with "perl-" and either omit the "::" or replace it with "-".

This section demonstrates the `yum` program, but any RPM-compatible search program will do.

Apply It

When searching, list all available packages as `yum list all`, and then pass the output through `grep` and find entries that begin with `perl-` as `| grep ^perl`.

TYPE THIS

```
yum list all | grep ^perl | grep -i date
```



RESULTS

YUM displays all Perl module packages. The `grep` parameter `-i` makes the search case-insensitive.

Search for Perl Modules in Red Hat Linux

- 1 Type `yum list all | grep ^perl`.

Note: Append `| less` to the command to page through the list.

YUM displays a list of available Perl modules.

```
adammm@localhost:~$ yum list all | grep ^perl
perl-1386                                4:5.8.8-10.el5_3.1    installed
perl-String-CRC32.i386                  1.4-2.fc6              installed
perl-Archive-Tar.noarch                 1.30-1.fc6             base
perl-Archive-Zip.noarch                 1.16-1.2.1             base
perl-Authen-RADIUS.noarch               0.13-1.el5.centos     extras
perl-BSD-Resource.i386                  1.28-1.fc6.1           base
perl-Bit-Vector.i386                    6.4-2.2.2.1            base
perl-Carp-Clan.noarch                   5.3-1.2.1              base
perl-Compress-Zlib.i386                  1.42-1.fc6             base
perl-Config-General.noarch              2.40-1.el5             base
perl-Convert-ASN1.noarch                 0.20-1.1               base
perl-Crypt-SSLeay.i386                   0.51-11.el5            base
perl-DBD-MySQL.i386                      3.0007-2.el5           bases
perl-DBD-Pg.i386                         1.49-2.el5_3.1         updates
perl-DBI.i386                            1.52-2.el5             base
perl-Data-Dump.noarch                    0.02-1.el5.centos     extras
perl-Data-Calc.i386                      5.4-1.2.2.1            base
perl-Date-Manip.noarch                   5.44-1.2.1             base
perl-Digest-HMAC.noarch                  1.01-15                 base
perl-Digest-SHA1.i386                    2.11-1.2.1             base
perl-Geo-IP.i386                         1.31-1.el5.centos     extras
perl-HTML-Parser.i386                    3.55-1.fc6             base
perl-HTML-Tagset.noarch                  3.10-2.1.1             base
perl-IO-Socket-INET6.noarch              2.51-2.fc6             base
perl-IO-Socket-SSL.noarch                1.01-1.fc6             base
perl-IO-String.noarch                    1.08-1.1.1             base
perl-IO-Zlib.noarch                      1.04-4.2.1             base
perl-LDAP.noarch                         1:0.33-3.fc6            base
perl-Mail-POP3Client.noarch              2.17-1.el5.centos     extras
perl-MailTools.noarch                    1.77-1.el5.centos     extras
perl-Mozilla-LDAP.i386                   1.5-2.4.el5            base
perl-NKF.i386                             2.07-1.1.fc6           base
perl-Net-DNS.i386                         0.59-3.el5             base
perl-Net-IMAP-Simple.noarch              1.17-1.el5.centos     extras
perl-Net-IMAP-Simple-SSL.noarch           1.3-1.el5.centos       extras
perl-Net-IP.noarch                       1.25-2.fc6             base
```

- 2 Type `yum info package`.

- The module's package name.
- The module's version.
- The module's summary.
- The module's description.

```
adammm@localhost:~$ yum info perl-Date-Manip
Loaded plugins: fastestmirror
Available Packages
Name      : perl-DateManip
Arch      : noarch
Version   : 5.44
Release   : 1.2.1
Size      : 144 k
Summary   :
URL       : http://search.cpan.org/dist/date-manip/
License   : GPL or Artistic
Description:
perl-Date-Manip is a Perl module for manipulating dates.

adammm@localhost:~$
```


Install Perl Modules in Red Hat Linux

It may seem redundant to search and install Perl modules as a Red Hat package rather than through CPAN; however, this pre-packaged method is preferred. The Red Hat maintainers strive to ensure all programs work as designed on a Red Hat system as they do anywhere else. If there is a quirk on Red Hat that warrants a source-code change in the module, but the change only applies to Red Hat-based systems, the RPM package will have the change but CPAN will not.

You can also easily upgrade and remove Red Hat packages. Red Hat packages also have inherent dependency checking. If a module depends on another third-party module or program, then the RPM module installation process properly downloads and installs any dependencies.

Apply It

If after installing a module you are not sure how to use it in a Perl script, you can use the `perldoc` program to open the module's manual. However, `perldoc` is not aware of DEB package names, only the names of Perl modules.

TYPE THIS

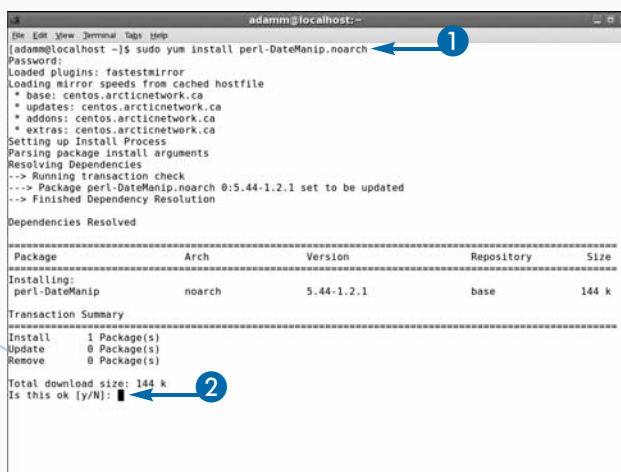
```
perldoc Date::Manip
```

RESULTS

The `Date::Manip` module's documentation displays.

Install Perl Modules in Red Hat Linux

- 1 Type **`sudo yum install package`**.
- 2 Press **y** to confirm the download and installation, if asked.



```
adammm@localhost:~$ sudo yum install perl-DateManip.noarch
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.arcticnetwork.ca
 * updates: centos.arcticnetwork.ca
 * addons: centos.arcticnetwork.ca
 * extras: centos.arcticnetwork.ca
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
--> Package perl-DateManip.noarch 0:5.44-1.2.1 set to be updated
--> Finished Dependency Resolution

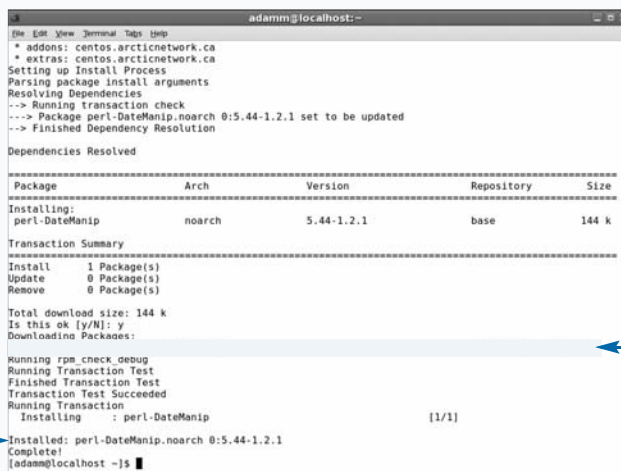
Dependencies Resolved

=====
Package                Arch      Version      Repository    Size
=====
Installing:
perl-DateManip          noarch    5.44-1.2.1    base          144 k

Transaction Summary
-----
Install  1 Package(s)
Update   0 Package(s)
Remove   0 Package(s)

Total download size: 144 k
Is this ok [y/N]:
```

- YUM downloads the Perl module package.
- YUM installs the Perl module package.



```
adammm@localhost:~$ sudo yum install perl-DateManip.noarch
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.arcticnetwork.ca
 * updates: centos.arcticnetwork.ca
 * addons: centos.arcticnetwork.ca
 * extras: centos.arcticnetwork.ca
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
--> Package perl-DateManip.noarch 0:5.44-1.2.1 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch      Version      Repository    Size
=====
Installing:
perl-DateManip          noarch    5.44-1.2.1    base          144 k

Transaction Summary
-----
Install  1 Package(s)
Update   0 Package(s)
Remove   0 Package(s)

Total download size: 144 k
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing      : perl-DateManip                               [1/1]
Installed: perl-DateManip.noarch 0:5.44-1.2.1
Complete!
adammm@localhost ~$
```

Search for and Download Perl Modules Manually

You can manually search and download a Perl module if your preferred installation package system does not have that module available, or if you want to see the module's source code. The Search CPAN Web site, located at <http://search.cpan.org>, has all of its modules from its database available.

Sometimes, a manual download may be your only option if the module's DEB, RPM, or PPM package cannot be found or is out of date, and you cannot install the module using the CPAN program directly. Also, if you want to make any changes to a Perl module, and examine how it works, you need to manually download its original archive package to access its complete source code. Even though Perl modules are already in "source code" format on your system, there are additional components that are included in the source archive that are not actually

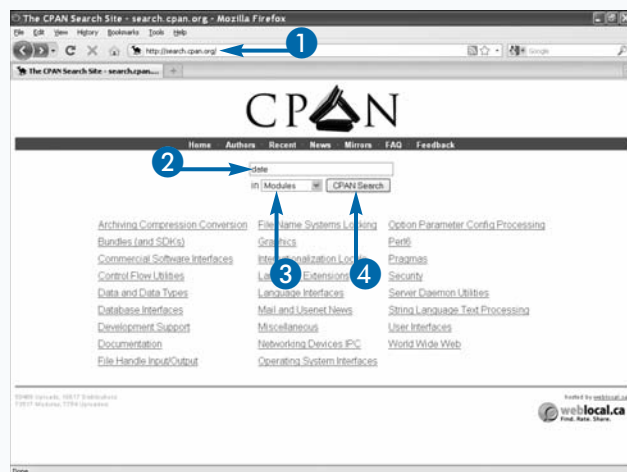
installed when you install the module. This process gives you a more "hands-on" approach to the Perl module building and installing process.

Be aware that manually downloading a module does not download its dependencies; if there are any, then you also need to manually download them. There is a dependency link on the module summary screen that allows you to review other required modules. Most times, the dependencies listed here are *core modules*, which ship with every distribution of Perl, and it is not necessary to download them again.

If you forget to download a dependency module, you can always search for and download it later. The build process, as described in the section, "Build and Install Perl Modules Manually," provides a list of everything that is missing on your system.

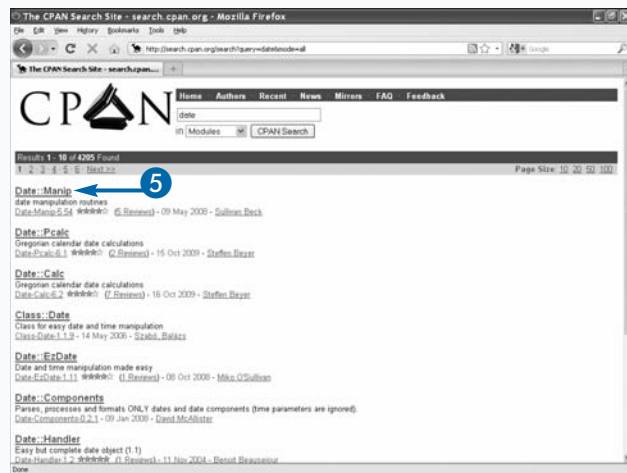
Search For and Download Perl Modules Manually

- 1 Type <http://search.cpan.org/> in your browser.
- 2 Type in a search term.
- 3 Select modules.
- 4 Click CPAN Search.



The CPAN search results page appears.

- 5 Click the name of the module you want to know about.

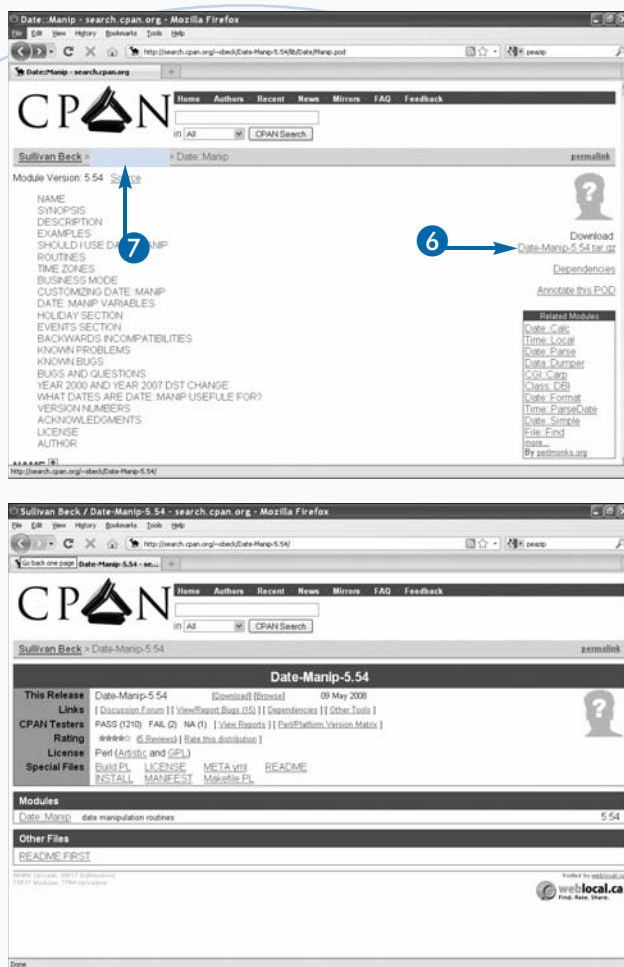


The module's manual appears.

- 6 Click the download link to download the module's `tar.gz` file.
- 7 Click the module's source summary link.

The module's version summary screen appears.

Note: You can also access the module's known bugs, license, install instructions, end-user reviews, and other documentation on this screen.



Apply It

Once the download is complete, uncompress the `tar.gz` archive. If you are downloading the module's source on Windows, you may need additional software to uncompress the `tar.gz` archive. The free programs 7-Zip, available from www.7-zip.org, and PeaZip, available from <http://peazip.sourceforge.net>, can handle this type of archive on Windows.

If you are downloading the source on Unix or Linux, use the following command in a terminal:

```
tar xfvz archive.tar.gz
```

This uncompresses the `tar.gz` contents under a new directory, usually with the same name as the archive itself plus its version number. The module is now ready to be built and installed.

The source code to the module consists of several other components. Each component is standardized according to the Perl documentation manual, `perldoc perlmodlib`.

If you are interested in creating a completely new module of your own, a useful learning technique is to examine the contents of an existing Perl module source. You can find the module's source code in a subdirectory called `lib`. You can find any test scripts in a subdirectory called `t`.

Build and Install Perl Modules Manually

You can manually build and install a Perl module if you are interested in knowing how it is actually built, compiled, and installed, or if you want to customize a module with your own code. This can be useful if a problem originates in an upstream module and you want to diagnose the bug, or if you want to add a new feature to that module.

You handle the actual build-and-install process using a few standard core utilities and programs that are available on all Linux distributions, and on Strawberry Perl on Windows. The ActiveState ActivePerl PPM provides these core programs as separate packages.

Again, it is important to stress that if you are using ActivePerl, Debian/Ubuntu, or Red Hat, you need to use the PPM, DEB, or RPM install methods.

The build-and-install process always uses the same four commands: `perl Makefile.PL`, `make`, `make test`, `sudo make install`.

The only variations are that Windows ActiveState and Strawberry Perl use `dmake`, not `make`, and Windows users do not require `sudo`.

After running the last command, the Perl module is installed in a location on your hard drive specific to locally installed modules. In Strawberry Perl, this is `c:\strawberry\perl\site\lib`, and in Linux, this is `/usr/local/share/perl/version`; however, your particular installation may vary. To force installation into a specific directory, add a `PREFIX` parameter to the first command:

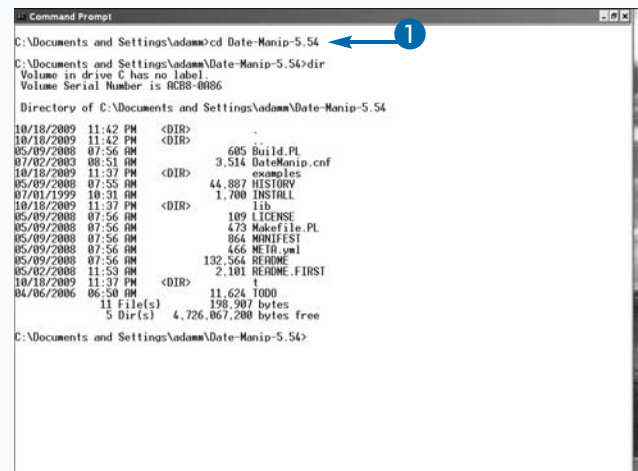
```
perl Makefile.PL PREFIX=DIRECTORY
```

The final installation directory uses that directory path.

Build and Install Perl Modules Manually

- 1 Go to the directory with the downloaded Perl module source code.

Uncompress the `tar.gz` archive if you have not done this already.



```
Command Prompt
C:\Documents and Settings\adam>cd Date-Manip-5.54
C:\Documents and Settings\adam\Date-Manip-5.54>dir
Volume in drive C: has no label
Volume Serial Number is BC88-0806

Directory of C:\Documents and Settings\adam\Date-Manip-5.54

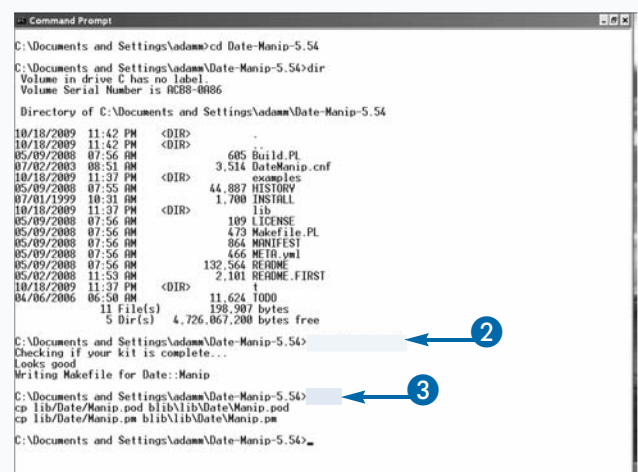
10/18/2009 11:42 PM <DIR> .
10/18/2009 11:42 PM <DIR> ..
05/09/2008 07:56 RM      605 Build.PL
07/02/2003 08:51 RM    3,514 DateManip.cnf
10/18/2009 11:37 PM      examples
05/09/2008 07:55 RM    44,887 HISTORY
07/01/1999 10:31 RM    1,700 INSTALL
10/18/2009 11:37 PM <DIR> lib
05/09/2008 07:56 RM    109 LICENSE
05/09/2008 07:56 RM    473 Makefile.PL
05/09/2008 07:56 RM    864 MANIFEST
05/09/2008 07:56 RM    466 META.yml
05/09/2008 07:56 RM   132,564 README
05/02/2008 11:53 RM    2,101 README.FIRST
10/18/2009 11:37 PM <DIR> t
04/06/2006 06:50 RM    11,624 TODO
                11 File(s)      198,907 bytes
                5 Dir(s)      4,726,067,200 bytes free

C:\Documents and Settings\adam\Date-Manip-5.54>
```

- 2 Type `perl Makefile.PL` and press Enter.
- 3 Type `make` (`dmake` on Windows) and press Enter.

The module build process begins.

Note: Remember, on Windows ActivePerl and Strawberry Perl, use `dmake` instead of `make` for all remaining commands.



```
Command Prompt
C:\Documents and Settings\adam>cd Date-Manip-5.54
C:\Documents and Settings\adam\Date-Manip-5.54>dir
Volume in drive C: has no label
Volume Serial Number is BC88-0806

Directory of C:\Documents and Settings\adam\Date-Manip-5.54

10/18/2009 11:42 PM <DIR> .
10/18/2009 11:42 PM <DIR> ..
05/09/2008 07:56 RM      605 Build.PL
07/02/2003 08:51 RM    3,514 DateManip.cnf
10/18/2009 11:37 PM      examples
05/09/2008 07:55 RM    44,887 HISTORY
07/01/1999 10:31 RM    1,700 INSTALL
10/18/2009 11:37 PM <DIR> lib
05/09/2008 07:56 RM    109 LICENSE
05/09/2008 07:56 RM    473 Makefile.PL
05/09/2008 07:56 RM    864 MANIFEST
05/09/2008 07:56 RM    466 META.yml
05/09/2008 07:56 RM   132,564 README
05/02/2008 11:53 RM    2,101 README.FIRST
10/18/2009 11:37 PM <DIR> t
04/06/2006 06:50 RM    11,624 TODO
                11 File(s)      198,907 bytes
                5 Dir(s)      4,726,067,200 bytes free

C:\Documents and Settings\adam\Date-Manip-5.54>perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for Date::Manip

C:\Documents and Settings\adam\Date-Manip-5.54>make
cp lib\Date\Manip.pod blib\lib\Date\Manip.pod
cp lib\Date\Manip.pm blib\lib\Date\Manip.pm

C:\Documents and Settings\adam\Date-Manip-5.54>
```

A message appears, letting you know that the build process was successful.

4 Type **make test** and press Enter.

The module's test suite begins.

- All module tests pass.

```

C:\Documents and Settings\adam\Date-Manip-5.54>
t/date_delta_2a.t ..... ok
t/date_delta_2b.t ..... ok
t/date_delta_russian.t ..... ok
t/date_delta_1.t ..... ok
t/date_delta_2a.t ..... ok
t/date_delta_2b.t ..... ok
t/date_delta_french.t ..... ok
t/date_delta_russian.t ..... ok
t/date_delta_sign.t ..... ok
t/date_french.t ..... ok
t/date_german.t ..... ok
t/date_misc_a.t ..... ok
t/date_misc_b.t ..... ok
t/date_romanian.t ..... ok
t/date_russian.t ..... ok
t/date_today_0.t ..... ok
t/date_today_1.t ..... ok
t/delta_a.t ..... ok
t/delta_b.t ..... ok
t/delta_delta_0.t ..... ok
t/delta_delta_1.t ..... ok
t/delta_delta_2a.t ..... ok
t/delta_delta_2b.t ..... ok
t/delta_format.t ..... ok
t/delta_romanian.t ..... ok
t/events.t ..... ok
t/getnext.t ..... ok
t/getprev.t ..... ok
t/normalize_business.t ..... ok
t/nthday.t ..... ok
t/pod.t ..... ok
t/pod_coverage.t ..... ok
t/recur_0.t ..... ok
t/recur_1.t ..... ok
t/settime.t ..... ok
t/unixdate.t ..... ok
All tests successful.
Files=40, tests=978, 8 wallclock secs ( 0.11 usr + 0.16 sys = 0.27 CPU)
Result: PASS
C:\Documents and Settings\adam\Date-Manip-5.54>

```

The test suite process is a success.

5 Type **sudo make install** (**dmake install** on Windows) and press Enter.

The module is now installed.

```

C:\Documents and Settings\adam\Date-Manip-5.54>
t/date_delta_2b.t ..... ok
t/date_delta_french.t ..... ok
t/date_delta_russian.t ..... ok
t/date_delta_sign.t ..... ok
t/date_french.t ..... ok
t/date_german.t ..... ok
t/date_misc_a.t ..... ok
t/date_misc_b.t ..... ok
t/date_romanian.t ..... ok
t/date_russian.t ..... ok
t/date_today_0.t ..... ok
t/date_today_1.t ..... ok
t/delta_a.t ..... ok
t/delta_b.t ..... ok
t/delta_delta_0.t ..... ok
t/delta_delta_1.t ..... ok
t/delta_delta_2a.t ..... ok
t/delta_delta_2b.t ..... ok
t/delta_format.t ..... ok
t/delta_romanian.t ..... ok
t/events.t ..... ok
t/getnext.t ..... ok
t/getprev.t ..... ok
t/normalize_business.t ..... ok
t/nthday.t ..... ok
t/pod.t ..... ok
t/pod_coverage.t ..... ok
t/recur_0.t ..... ok
t/recur_1.t ..... ok
t/settime.t ..... ok
t/unixdate.t ..... ok
All tests successful.
Files=40, tests=978, 8 wallclock secs ( 0.13 usr + 0.13 sys = 0.25 CPU)
Result: PASS
C:\Documents and Settings\adam\Date-Manip-5.54>dmake install
Installing C:\strawberry\perl\site\lib\Date-Manip.pm
Installing C:\strawberry\perl\site\lib\Date-Manip.pod
Appending installation info to C:\strawberry\perl\lib\perllocal.pod
C:\Documents and Settings\adam\Date-Manip-5.54>

```

Extra

If something goes wrong, an error is printed in `make`'s output. Usually this is as simple as a missing or out-of-date module. Although rare, if it is a syntax issue, try to locate a more recent version of the module, and try upgrading your local copy of Perl.

When installing an upgraded version of a Perl module, you use the exact same steps. The `make install` step is usually smart enough to identify if a module already exists and will install a new copy over the old copy.

Uninstalling modules, just like with CPAN, is not exactly possible. You could use the CPANPLUS program as described earlier in this chapter, but if that is not available, you have to search for the module files directly. Usually, modules named `Class::Package` are found in the `Class` directory, as a file called `Package.pm`. So, in this example, `Date::Manip` is stored as `Date/Manip.pm` somewhere on your hard drive.

Actually, *somewhere* is overly vague; the module is installed in one of your `@INC` directories. Type the command `perl -V` to see a list of the Perl `@INC` directories, relevant to your Perl distribution.

Introducing the Apache CGI Handler

The Apache Common Gateway Interface (or CGI) handler is the core component in Apache that links incoming HTTP requests from users to the CGI Perl scripts you have developed.

There are several implementations of CGI in Apache, and you can dynamically install each one into an Apache server using a *shared-object* file. The simplest interface, *mod_cgi*, provides the most rudimentary conduit for launching an instance of Perl for each CGI request.

Apache Configuration

Before you can enable CGI, you need to make some global configurations on the Apache server.

Enable the CGI Module

Apache ships with a module, *mod_cgi.so*, that provides CGI functionality. It must be enabled before any Perl CGI scripts will work. On most installations of Apache, the CGI module is enabled by default.

```
LoadModule cgi_module modules/mod_cgi.so
```

This directive is an example; the exact path to *mod_cgi.so* may be different on your system, but as long as the *LoadModule* directive exists somewhere, the CGI module should be activated.

Enable the CGI Handler

You require a special configuration command in the Apache *httpd.conf* file. This instructs Apache to execute Perl scripts as CGI scripts.

```
AddHandler cgi-script .pl
```

You can technically place this command anywhere inside of *httpd.conf*, or in any of its support files.

Restart Apache

If either the CGI module or the CGI handler was not previously enabled, and you have to manually add these directives in, then you must restart the Apache Web server.

For more information on restarting the Apache Web server on Windows, see Chapter 4. For more information about Linux, see Chapter 5.

More complex interfaces, such as *mod_perl*, actually embed the Perl interpreter directly into Apache, this increases the speed of Apache and Perl when serving CGI requests. For more information about *mod_perl*, see Chapter 23.

The core of this chapter will concern enabling the simpler CGI handler module, *mod_cgi*.

Using the CGI Handler

You must activate the CGI handler onto a particular Web server directory or Web site URI. This establishes a particular directory where all files contained within it will be considered CGI scripts.

Using the Handler on a Global URI

The *ScriptAlias* directive instructs Apache to assign a site-wide URI directory, typically */cgi-bin/*, to a specific directory on the Web server. All files found within this directory are passed through the CGI handler and executed as CGI scripts.

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
```

This means that the same CGI scripts are available on all virtual domains at the URI */cgi-bin/*.

You must assign the *ScriptAlias* directive in the Apache core configuration file. All Web sites and domains hosted by the server will honor the single directory and path.

Using the Handler on a Particular Directory

You may grant an individual directory access to the CGI handler through the *Options* directive. This is useful if you want to enable the CGI handler for an individual user or virtual domain, but not globally at a specific URI.

The *Options* directive may exist within a *<directory>*, *<location>*, *<files>*, or *<proxy>* configuration section to limit exposure to the CGI handler.

```
<directory /var/www/testsite/cgi-bin/>  
    Options +ExecCGI  
</directory>
```

You must assign the *Options* directive that has one configuration section in the Apache core configuration file.

Using the CGI Handler *(continued)*

Using the Handler in an .htaccess File

The `Options` directive is also legal within an `.htaccess` file saved within a particular directory. The configuration section becomes redundant, as all requests going to that directory now honor the directive.

This is useful if you want to grant users access to create their own custom `cgi-bin` directory, without granting them access to the core Apache configuration file.

For this to work, you must create an `AllowOverride` directive within a `<directory>` configuration section in the configuration file.

Using Perl for Apache CGI

You need to prepare your Perl scripts before they can work with the Apache CGI handler.

Perl Files

You must cleanly execute the Perl files themselves from the command-line. If the command `perl myscript.pl` produces an error on the command-line, it generates a similar error in the Apache logs.

Also, on Unix systems only, you must set the Perl file's execution bit. This ensures that Apache understands this file is a dynamic script that it must execute, not a static file that it should simply read.

Perl Interpreter Path Header

The first line of the Perl script must be the path to the Perl binary. As you learned in Chapter 6, the header must be the correct path.

It may be acceptable for a command-line-only Perl script in Windows to have a faulty header; Windows internally associates the `.pl` extension to a particular executable file. However, this type of association has no bearing on Apache and must be properly defined.

PERL DISTRIBUTION	DEFAULT HEADER PATH
Any Linux Perl	<code>#!/usr/bin/perl</code>
Windows ActiveState Perl	<code>#!C:\Perl\bin\perl.exe</code>
Windows Strawberry Perl	<code>#!C:\strawberry\perl\bin\perl.exe</code>

Perl Output

All content printed by Perl on *standard output*, or `STDOUT`, goes back to Apache, which relays the text to the user's browser. Any content printed on *standard error*, or `STDERR`, is relayed to the Apache error log.

According to the HTTP protocol standard, the Web server must send all page requests to the browser in two parts: the HTTP header, which introduces the content, followed by the actual content. The header consists of multiple lines, each with one HTTP header field and definition. The last HTTP header signals that the content is coming next by adding a blank line after its own.

Apache sends some HTTP headers automatically, but it leaves the door open for the CGI script to provide more. This allows the Perl script to send cookies, manipulate the HTTP session state, or provide other advanced headers according to the HTTP/1.1 specs.

At a minimum, Perl must provide at least one new header, *content-type*, prior to any actual content. The content-type supplies the MIME type, which describes the actual content. Often this is represented as

```
print "Content-type: text/html\n\n";
```

Note the two carriage-returns (`\n\n`); the browser now treats all printed data as HTML code.

The content that Perl is printing does not have to be HTML code. It could be raw image data, such as a JPEG. The only requirement is that the content be prefixed by the correct MIME type in the content-type header — in this case, `Content-type: image/jpeg`.

Create a User Directory for Apache in Windows

Creating a user directory for Apache is a quick way to create a development environment for your first Web pages. Even if your system only has one user, following this method means you can easily create a unique URL that is specific to your own project. Once you set it up, you can only place static HTML files and images in this folder, but later you will be able to save Perl scripts in a subdirectory and execute them through a Web browser. This process involves editing the core Apache configuration file, typically named `httpd.conf`, and enabling the `mod_userdir.so` module. You must also enable an additional configuration file, `httpd-userdir.conf`, which is unique to Apache on Windows. Once you have completed this, you must restart the Web server.

The default location for user home directories in Apache in Windows is `C:\Documents and Settings\userid\`

Create a User Directory for Apache in Windows

- 1 Open the Apache configuration file in a text editor.

Note: For information on opening the Apache configuration file, see Chapter 4.

My Documents\My Webpage\.. You need to create this path manually for each user on your system that requires a user Web directory in Apache. It is possible to customize this directory by editing the `UserDir` directive and directory section in `httpd-userdir.conf`.

Once you have configured and restarted Apache, and the directory exists, you can access all files created on this directory from a browser at the address `http://localhost/~userid/`. The user's directory name, `~userid`, is a throwback to the Unix heritage of Apache and is not a technical requirement on Windows.

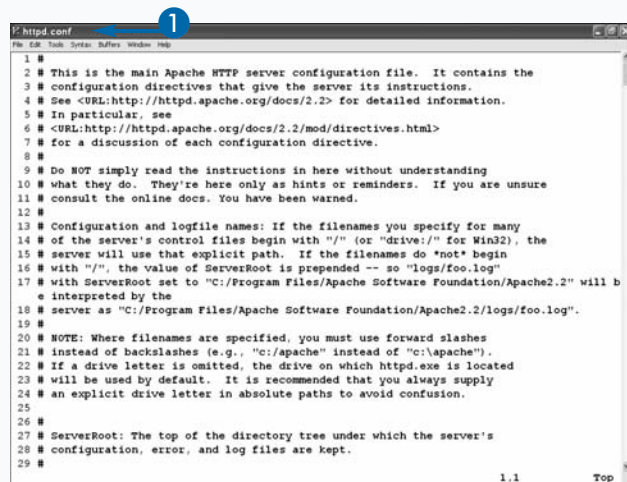
If you do not like this format, you can always manually create a directory on the server from the root of the default domain. You can access files that you store in this directory under the server's default root URL as `http://localhost/userdir/`.

- 2 Search for the text `mod_userdir.so`.

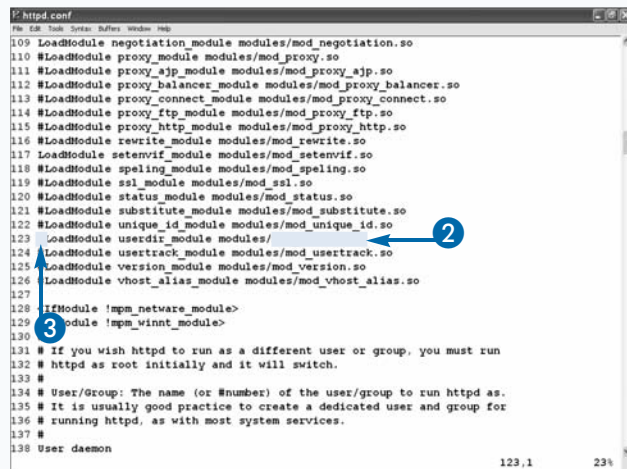
Note: In Notepad, click `Edit → Find`. Type `mod_userdir.so` and click `Find Next`.

- 3 Remove the hash symbol (#) to uncomment the `LoadModule` directive.

Note: If you cannot find the `LoadModule` directive that imports `mod_userdir.so`, then you need to manually type it in.



```
1 #
2 # This is the main Apache HTTP server configuration file. It contains the
3 # configuration directives that give the server its instructions.
4 # See <URL:http://httpd.apache.org/docs/2.2> for detailed information.
5 # In particular, see
6 # <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
7 # for a discussion of each configuration directive.
8 #
9 # Do NOT simply read the instructions in here without understanding
10 # what they do. They're here only as hints or reminders. If you are unsure
11 # consult the online docs. You have been warned.
12 #
13 # Configuration and logfile names: If the filenames you specify for many
14 # of the server's control files begin with "/" (or "drive:/" for Win32), the
15 # server will use that explicit path. If the filenames do "not" begin
16 # with "/", the value of ServerRoot is prepended -- so "logs/foo.log"
17 # with ServerRoot set to "C:/Program Files/Apache Software Foundation/Apache2.2" will be
18 # interpreted by the
19 # server as "C:/Program Files/Apache Software Foundation/Apache2.2/logs/foo.log".
20 #
21 # NOTE: Where filenames are specified, you must use forward slashes
22 # instead of backslashes (e.g., "c:/apache" instead of "c:\apache").
23 # If a drive letter is omitted, the drive on which httpd.exe is located
24 # will be used by default. It is recommended that you always supply
25 # an explicit drive letter in absolute paths to avoid confusion.
26 #
27 # ServerRoot: The top of the directory tree under which the server's
28 # configuration, error, and log files are kept.
29 #
```



```
109 LoadModule negotiation_module modules/mod_negotiation.so
110 #LoadModule proxy_module modules/mod_proxy.so
111 #LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
112 #LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
113 #LoadModule proxy_connect_module modules/mod_proxy_connect.so
114 #LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
115 #LoadModule proxy_http_module modules/mod_proxy_http.so
116 #LoadModule rewrite_module modules/mod_rewrite.so
117 #LoadModule setenvif_module modules/mod_setenvif.so
118 #LoadModule speling_module modules/mod_speling.so
119 #LoadModule ssl_module modules/mod_ssl.so
120 #LoadModule status_module modules/mod_status.so
121 #LoadModule substitute_module modules/mod_substitute.so
122 #LoadModule unique_id_module modules/mod_unique_id.so
123 LoadModule userdir_module modules/
124 #LoadModule usertrack_module modules/mod_usertrack.so
125 #LoadModule version_module modules/mod_version.so
126 #LoadModule vhost_alias_module modules/mod_vhost_alias.so
127
128 #IfModule !mpm_netware_module>
129 #Module !mpm_winnt_module>
130
131 # If you wish httpd to run as a different user or group, you must run
132 # httpd as root initially and it will switch.
133 #
134 # User/Group: The name (or #number) of the user/group to run httpd as.
135 # It is usually good practice to create a dedicated user and group for
136 # running httpd, as with most system services.
137 #
138 User daemon
```

- 4 Search for the text `httpd-userdir.conf`.
- 5 Remove the hash symbol (`#`) to uncomment the `Include` directive.
- 6 Save the Apache configuration file.

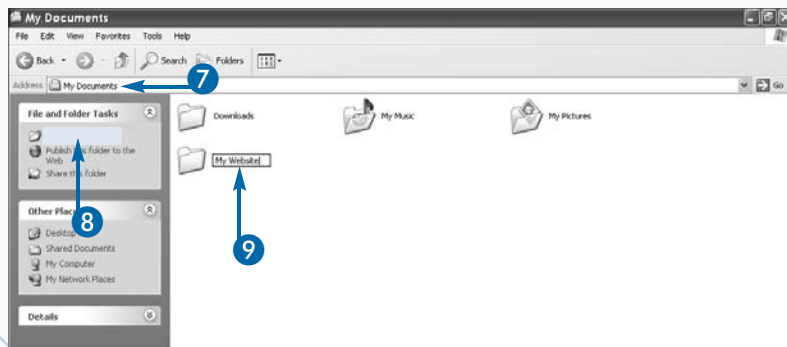
```

441
442 # Server-pool management (MPM specific)
443 #Include conf/extra/httpd-mpm.conf
444
445 # Multi-language error messages
446 #Include conf/extra/httpd-multilang-errordoc.conf
447
448 # Fancy directory listings
449 #Include conf/extra/httpd-autoindex.conf
450
451 # Language settings
452 #Include conf/extra/httpd-languages.conf
453
454 # User home directories
455 Include conf/extra/httpd-userdir.conf
456
457 # Real-time info on requests and configuration
458 #Include conf/extra/httpd-info.conf
459
460 # Virtual hosts
461 #Include conf/extra/httpd-vhosts.conf
462
463 # Local access to the Apache HTTP Server Manual
464 #Include conf/extra/httpd-manual.conf
465
466 # Distributed authoring and versioning (WebDAV)
467 #Include conf/extra/httpd-dav.conf
468
469 # Various default settings
470 #Include conf/extra/httpd-default.conf

```

- 7 Open your My Documents folder.
- 8 Click Make a new folder.
- 9 Type **My Website** and press Enter.
- 10 Restart Apache on Windows.

Note: For more information, about starting and stopping Apache, see Chapter 4.



Apply It

The Apache user directory configuration is stored in the file `C:\Program Files\Apache Software Foundation\Apache2.2\conf\extra\httpd-userdir.conf`. This is the file that is included in the main `httpd.conf` file in step 4. Editing this file allows you to customize the default subdirectory name under each user's home directory, `My Website`, as well as the default directory permissions that are applied when you access this path from a Web browser.

Create a new HTML file called `index.html` in the `My Website` directory, and insert some dummy text. When a browser requests a URL, and no filename is specifically requested, Apache defaults to `index.html`; this is a good way to make sure that everything is working.

Once you create the file, open your Web browser to the address `http://localhost/~userid/`. You should see the text you saved in the static HTML file.

If you receive a "403 Forbidden" error message in your browser, it may mean that the `My Website` directory does not exist, or it does not contain any files.

If you still cannot access your user Web directory, try monitoring the Apache error log. For information on how to do this, see the section, "Read the Apache Logs."

Create a User Directory for Apache in Linux

Creating a user directory for Apache is a quick way to create a development environment for your Web pages. By default, you can only place static HTML and images in this folder, but later you will be able to save Perl scripts in this directory and execute them through a Web browser.

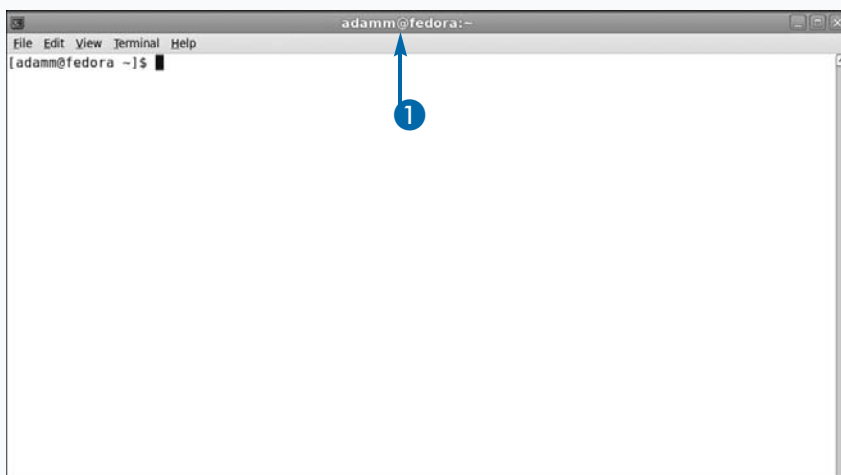
Depending on the version of Linux you are using, there is a slight difference in how you enable the `userdir` module. If you are using Linux based upon Debian or Ubuntu, the `userdir` module and configuration is available through the `mods-available` infrastructure, and you must specifically enable it through the program `a2enmod`. If you are using a Red Hat-based version of Linux, you may need to manually edit the `httpd.conf` configuration file, or, in the case of Mandriva Linux, install the package `apache-mod_userdir`. The default location for user

home directories in Apache in Linux is `/home/userid/public_html/`. You need to create this path manually for each user on your system that requires a user Web directory in Apache.

Once you have configured and restarted Apache, and the directory exists, you can access all files created on this directory from a browser at the address `http://localhost/~userid/`. If you do not like the tilde-`userid` format, you can always manually create a directory on the server from the root of the default domain. Instead of enabling the `userdir` module as described in this section, just create a new directory under the root URL. In Debian or Ubuntu, create a new directory under `/var/www/userdir`. In Red Hat, use `/var/lib/htdocs/userdir`. You can access files that you store in this directory under the server's default root URL at `http://localhost/userdir/`. You need to set ownerships of this subdirectory to the user.

Create a User Directory for Apache in Linux

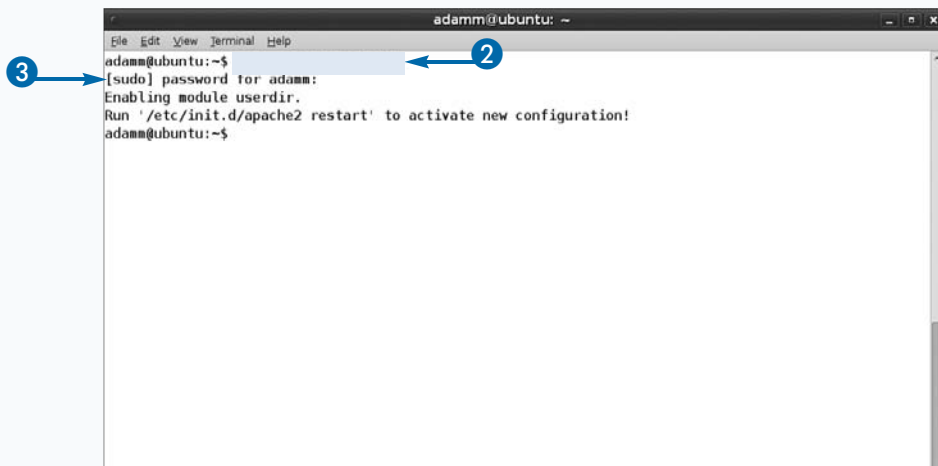
- 1 Open a Terminal window in Linux.



- 2 (Debian/Ubuntu Linux only) Type **sudo a2enmod userdir** and press Enter.

(Mandriva Linux only) As root, type **gurpmi apache-mod_userdir** and press Enter.

- 3 If prompted, type in your password and press Enter.



Note: If you used the Debian, Ubuntu, or Mandriva steps, skip to step 9.

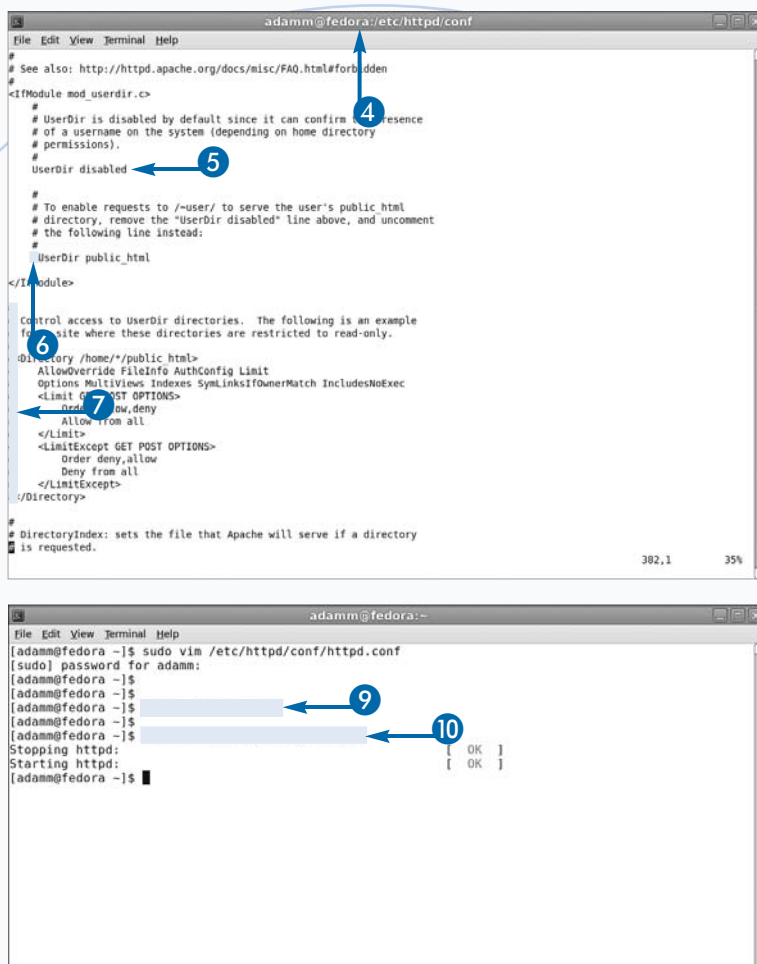
- 4 (Red Hat Linux only) Open the Apache configuration file in a text editor.

Note: For information on opening the Apache configuration file, see Chapter 5.

- 5 Comment out the line `UserDir disabled`.
- 6 Uncomment the line `UserDir public_html`.
- 7 Uncomment the block that “controls access to UserDir directories.”
- 8 Save the `httpd.conf` file.

- 9 Type `mkdir ~/public_html` and press Enter.
- 10 Restart Apache on Linux.

Note: For more information about starting and stopping Apache, see Chapter 5.



Apply It

Create a new HTML file called `index.html` in the `public_html` directory, and insert some dummy text. When a browser requests a URL, and no filename is specifically requested, Apache defaults to `index.html`; this is a good way to make sure that everything is working.

Once you create the file, open your Web browser to the address `http://localhost/~userid/`. You should see the text you saved in the static HTML file.

If you receive a “403 Forbidden” error message in your browser, it may mean the permissions are not set up correctly on the user’s home directory or the `public_html` directory, or that no actual files exist. If this is the case, at a minimum, you must make the user’s home directory *world-readable* and the Web directory *world-readable* and *world-executable*.

```
chmod 0711 ~userid
chmod 0755 ~userid/public_html
```

If you still cannot access your user Web directory, try monitoring the Apache error log. For information on how to do this, see the section, “Read the Apache Logs.”

Enable the Apache CGI Module and Handler

Enabling the CGI module and handler in Apache is one of the steps required in order to execute Perl scripts in a Web browser and create a dynamic Web page. After enabling the module and handler, you must configure a directory to use the handler, and then Apache can execute all Perl scripts placed in this directory.

You enable the module and the handler in the Apache configuration file. The exact configuration file depends on your operating system. If you are using Windows, edit the file `Apache Install Dir\conf\httpd.conf`. If you are on Debian or Ubuntu Linux, edit `/etc/apache2/mods-enabled/mime.conf`. For Red Hat-based Linux, edit `/etc/httpd/conf/httpd.conf`.

Because this is a two-step process, some operating systems require two changes in one file; others require

that you run a command and make one change in a file. On Windows- and Red Hat-based systems (including Mandriva and Fedora), edit the `httpd.conf` configuration file previously identified. If you are using Debian or Ubuntu Linux, you can enable the CGI module with a command, `a2enmod`, and then configure the CGI handler in the `mime.conf` file.

Once the CGI module and handler are online, you need to grant some sort of directory-level access to the handler. You can do this globally by using the `ScriptAlias` directive, which binds a global URI path for the entire server to the CGI handler. If you want to grant CGI access to a specific user or Web domain, see the next section.

Apache ships with a pre-configured `ScriptAlias` directory, `http://localhost/cgi-bin/`, which points to a `cgi-bin` folder on the server.

Enable the Apache CGI Module and Handler

- 1 Open the Apache configuration file containing the CGI module and handler in a text editor.

Note: If you are using Debian or Ubuntu Linux, skip to step 4.

- 2 Search the file for the directive `LoadModule cgi_module`.
- 3 Remove the hash (#) to uncomment the directive, if it is commented out.

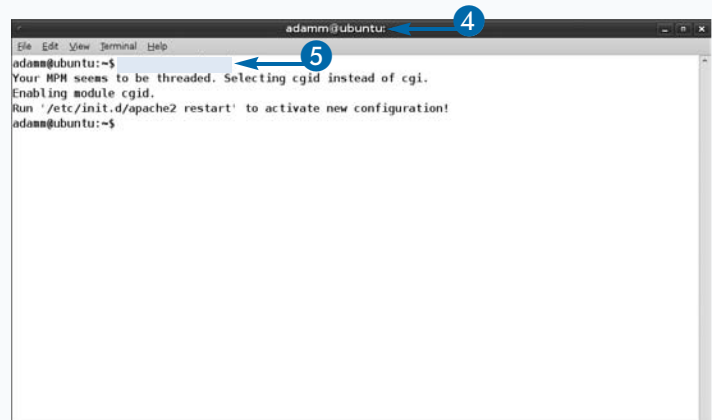


```
67 #LoadModule authn_dbd_module modules/mod_authn_dbd.so
68 #LoadModule authn_dbm_module modules/mod_authn_dbm.so
69 LoadModule authn_default_module modules/mod_authn_default.so
70 LoadModule authn_file_module modules/mod_authn_file.so
71 #LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
72 #LoadModule authz_dbm_module modules/mod_authz_dbm.so
73 LoadModule authz_default_module modules/mod_authz_default.so
74 LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
75 #LoadModule authz_host_module modules/mod_authz_host.so
76 #LoadModule authz_owner_module modules/mod_authz_owner.so
77 LoadModule authz_user_module modules/mod_authz_user.so
78 LoadModule autoindex_module modules/mod_autoindex.so
79 LoadModule cache_module modules/mod_cache.so
80 LoadModule cern_meta_module modules/mod_cern_meta
81 LoadModule cgi_module modules/mod_cgi.so
82 LoadModule charset_lite_module modules/mod_charset_lite.so
83 #LoadModule dav_module modules/mod_dav.so
84 #LoadModule dav_fs_module modules/mod_dav_fs.so
85 #LoadModule dav_lock_module modules/mod_dav_lock.so
86 #LoadModule dbd_module modules/mod_dbd.so
87 #LoadModule deflate_module modules/mod_deflate.so
88 LoadModule dir_module modules/mod_dir.so
89 #LoadModule disk_cache_module modules/mod_disk_cache.so
90 #LoadModule dumpio_module modules/mod_dumpio.so
91 LoadModule env_module modules/mod_env.so
92 #LoadModule expires_module modules/mod_expires.so
93 #LoadModule ext_filter_module modules/mod_ext_filter.so
94 #LoadModule file_cache_module modules/mod_file_cache.so
95 #LoadModule filter_module modules/mod_filter.so
96 #LoadModule headers_module modules/mod_headers.so
```

Note: Steps 4 and 5 are for Debian/Ubuntu Linux only.

- 4 Open a Terminal window.
- 5 Type `sudo a2enmod cgi` and press Enter.

Note: It is okay if Apache selects the `cgid` module instead of `cgi`, if it is supported by your system.



```
adam@ubuntu:~$ sudo a2enmod cgi
Your NPM seems to be threaded. Selecting cgid instead of cgi.
Enabling module cgid.
Run '/etc/init.d/apache2 restart' to activate new configuration!
adam@ubuntu:~$
```


- 6 Search the file for the directive `AddHandler cgi-script`.
- 7 Remove the hash (#) to uncomment the directive.
- 8 Type `.pl` at the end of the directive.
- 9 Save the configuration file.

```

377 # If the AddEncoding directives above are commented-out, then you
378 # probably should define those extensions to indicate media types:
379 #
380 AddType application/x-compress .Z
381 AddType application/x-gzip .gz .tgz
382
383 #
384 # AddHandler allows you to map certain file extensions to "handlers":
385 # extensions unrelated to filetype. These can be either built into the server
386 # or added with the Action directive (see below)
387 #
388 # To use CGI scripts outside of ScriptAliased directories:
389 # (You will also need to add "ExecCGI" to the "Options" directive.)
390 #
391 # AddHandler cgi-script .cgi
392
393 # For type maps (negotiated resources):
394 #AddHandler type-map var
395
396 #
397 # Filters allow you to process content before it is sent to the client.
398 #
399 # To parse .shtml files for server-side includes (SSI):
400 # (You will also need to add "Includes" to the "Options" directive.)
401 #
402 #AddType text/html .shtml
403 #AddOutputFilter INCLUDES .shtml
404 </IfModule>
405
406 #
  
```

- 10 Restart the Apache service.

Note: On Windows, the `net stop` and `net start` commands are a shortcut to restarting Apache on the command-line.

```

C:\Program Files\Apache Software Foundation\Apache2.2>net stop httpd.conf
C:\Program Files\Apache Software Foundation\Apache2.2>
The Apache2.2 service was stopped successfully.

C:\Program Files\Apache Software Foundation\Apache2.2>
C:\Program Files\Apache Software Foundation\Apache2.2>net start
The Apache2.2 service is starting.
The Apache2.2 service was started successfully.

C:\Program Files\Apache Software Foundation\Apache2.2>
  
```

Apply It

Apache ships with a sample Perl script in its global CGI directory called `printenv.pl`. This script can be extremely useful as you use it to report back the Apache server's environment variables. You can also use it to validate that the CGI module and handler are enabled correctly. Once you have restarted Apache, go to the following URL in a Web browser.

```
http://localhost/cgi-bin/printenv.pl
```

You should see a listing of environment variables and their values. Depending on your Perl installation, you may need to manually edit the `printenv.pl` file. Change the first line to specify the path to the Perl interpreter. For example, if you are using the Strawberry Perl distribution, use the following header:

```
#!C:\Strawberry\Perl\bin\perl.exe
```

The output of `printenv.pl` is actually very close to the output of the `set` command on Windows and Linux. In fact, it is the exact same idea. The only difference is that this is the environment from the perspective of the Apache server and the user's Web browser. In Perl, all environment variables are pre-populated into the hash variable `%ENV`. This can be very useful to pull information about the user's session into your Perl CGI scripts:

```
print "Hello, $ENV{ 'REMOTE_ADDR' }.<p>";
print "You're using the browser: $ENV{ 'HTTP_USER_AGENT' }.<p>";
```

Configure a Directory to Use the CGI Handler

Once the user's personal Web directory is active, and you have enabled both the CGI module and handler, the final step is to configure a directory to use the CGI handler. This is usually a subdirectory called `cgi-bin`, but you can customize its name. Apache executes Perl scripts that are placed in this new directory through the CGI handler, allowing you to produce dynamic Web pages for users. After you create the folder, you need to configure Apache to use the folder. You need to open the Apache configuration file and edit the `<Directory>` configuration section; you must append its `AllowOverride` directive with the value `Options`. Then, within the new `cgi-bin` directory, you must create a special file called `.htaccess`. It only contains one line, `Options +ExecCGI`. The location of the Apache configuration file depends on your operating system. If

you are using Windows, edit `Apache Install Dir\conf\httpd.conf`. If you are on Debian- or Ubuntu-based Linux, edit `/etc/apache2/mods-enabled/userdir.conf`. For Mandriva Linux, edit `/etc/httpd/modules.d/67_mod_userdir.conf`. For all other Red Hat-based Linux distributions, edit `/etc/httpd/conf/httpd.conf`.

Apache comes with a pre-configured *global* CGI directory, as defined by the `ScriptAlias` directive, at the URL `http://localhost/cgi-bin/`. On a Windows system, this points to the directory `Apache Install Dir\cgi-bin\`. On Linux, it is either `/var/www/cgi-bin/` or `/usr/lib/cgi-bin/`. Files placed in this directory are automatically granted access to the Apache CGI handler. It is far better to create a new directory under your personal Web site URL, and grant it access to the CGI handler so that your CGI scripts run as unprivileged, regular users.

Configure a Directory to Use the CGI Handler

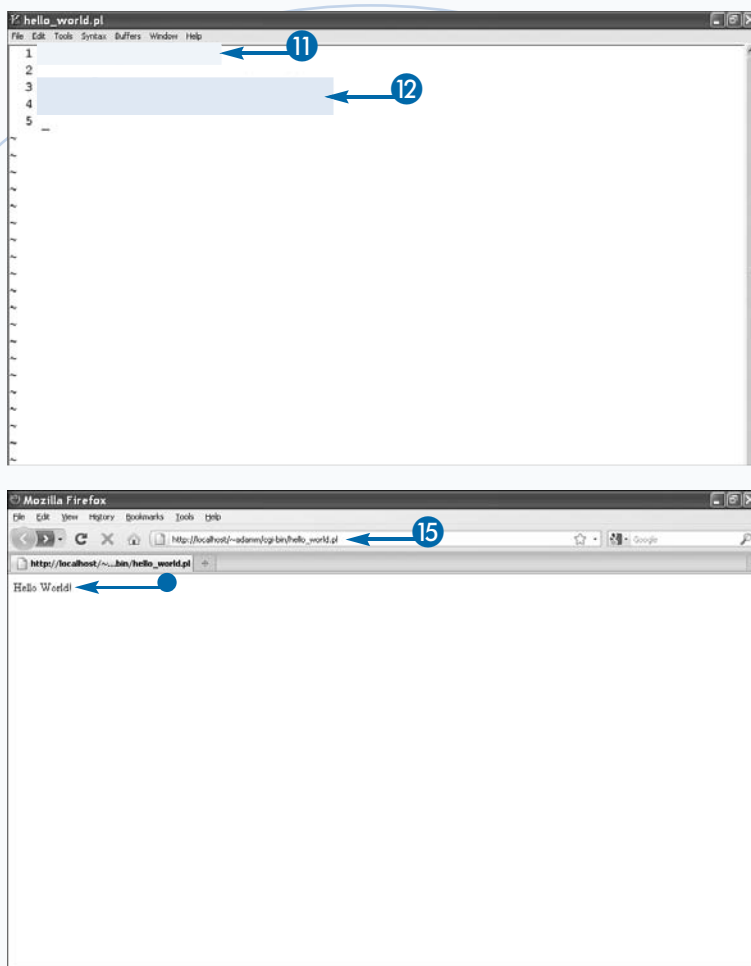
- 1 Open the Apache UserDir configuration file in a text editor.
- 2 Locate the `<Directory>` configuration section for UserDir directories.
- 3 Type **Options** as a new value in the `AllowOverride` directive.
- 4 Save the UserDir configuration file and exit the editor.
- 5 Restart Apache.
- 6 In a terminal, go to the My Website directory (`public_html` in Linux).
- 7 Type `mkdir cgi-bin` and press Enter.
- 8 Type `cd cgi-bin` and press Enter.
- 9 Type `echo Options +ExecCGI > .htaccess` and press Enter.

Note: Directives saved as `.htaccess` files do not require you to restart Apache.

```
1 # Settings for user home directories
2 #
3 # Required module: mod_userdir
4 #
5 #
6 # UserDir: The name of the directory that is appended onto a user's home
7 # directory if a ~user request is received. Note that you must also set
8 # the default access control for these directories, as in the example below.
9 #
10 UserDir "My Documents/My Website"
11
12 #
13 # Control access to UserDir directories. The following is an example
14 # for a site where these directories are restricted to read-only.
15 #
16
17 AllowOverride FileInfo AuthConfig Limit Indexes
18 Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
19 <Limit GET POST OPTIONS>
20   Order allow,deny
21   Allow from all
22 </Limit>
23 <LimitExcept GET POST OPTIONS>
24   Order deny,allow
25   Deny from all
26 </LimitExcept>
27 </Directory>
28
```

```
C:\Documents and Settings\adam>
C:\Documents and Settings\adam\My Documents\My Website>
C:\Documents and Settings\adam\My Documents\My Website>
C:\Documents and Settings\adam\My Documents\My Website\cgi-bin>
C:\Documents and Settings\adam\My Documents\My Website\cgi-bin>
```

- 10 Create a new Perl CGI script called `hello_world.pl`.
- 11 Insert the Perl interpreter path header.
- 12 Insert some static CGI text.
- 13 Save the script.
- 14 (Unix-only) On a command-line, type **`chmod +x hello_world.pl`** and press Enter.
- 15 In a browser, go to the Perl CGI script's URL.
 - The browser displays the output of the Perl CGI.



Apply It

The first line of the Perl CGI script is paramount. It must be the path to the Perl interpreter on your server.

PERL DISTRIBUTION	PERL BINARY PATH HEADER
Any Linux Perl	<code>#!/usr/bin/perl</code>
Windows ActiveState Perl	<code>#!C:\Perl\bin\perl.exe</code>
Windows Strawberry Perl	<code>#!C:\strawberry\perl\bin\perl.exe</code>

If you are on Linux, remember to set the script's executable-bit, as described in Chapter 6. You must do this once for every Perl script in your `cgi-bin` directory.

It is technically possible to make your entire personal Web directory executable, and not create a specific directory for CGI scripts at all. Just create the `.htaccess` file directly inside of the directory `My Websites` (`public_html` on Linux), as previously described.

In fact, you could eliminate the need for an `.htaccess` file entirely by appending `ExecCGI` to the `Options` directive in the `UserDir <Directory>` configuration section. However, do not do this unless you *explicitly trust* every user who has access to your system. Appending `ExecCGI` directly in the configuration file in this fashion grants every user's Web directory full access to the CGI handler by default.

Understanding the Apache Logs

All Web activity that Apache handles is stored in log files on the server. The standard access log stores information about every HTTP request, requester, the time, and the results. The standard error

log stores information about every unexpected failure, warning, and anomaly that happens on the server; this includes any errors induced by Perl CGI scripts.

Accessing the Apache Logs

You can read the Apache logs by opening the log file directly in a text editor. While this accesses the file, you need to close and re-open the file in order to see new content. On Linux, you can use the command `tail -f logfile` to stream activity in a terminal in real-time.

The Log Directory

The log directory into which Apache stores its logs differs, depending on your operating system.

OPERATING SYSTEM	APACHE LOG DIRECTORY
Windows	Apache Install Dir\logs
Debian/Ubuntu Linux	/var/log/apache2/
Red Hat Linux	/var/log/httpd/

The Activity Log

The activity log is saved in the log directory as `access.log` (or `access_log`). It describes all Web-based transactions the Apache server has responded to.

```
remote_host - - [timestamp] "GET URL HTTP/1.1"  
result bytes
```

In this format, an activity line describes the remote host's IP address or hostname, the remote log name (as a dash if not available), the remote username (as a dash if not available), the timestamp of the request, the actual HTTP request, the status results code of the request, and the number of bytes transferred. This represents the Apache *common* log format. Apache also provides a *combined* log format which extends the common format by adding the *referring* URL and user-agent environment values. The status result code follows RFC 2616. Typically, 200 means success, 403 is a "permission denied" error, 404 is a "file not found" error, and 500 is a server error. You can find the full list of status codes online at www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

The Error Log

The error log is saved in the log directory as `error.log` (or `error_log`). It describes any error messages triggered by any failed Web-based transactions the Apache server could not respond to.

```
[timestamp] [type] [client remote_host] Error  
message text
```

In this format, an error line describes the timestamp of the request, the type of failure (error, warning, or info), the remote host's IP address or hostname, and the raw error text describing the failure.

The actual text contained in the error log is *free form*, meaning that there is no set formatting standard that is followed. The error message text is usually generated by the program being executed. Typically, if a Perl CGI script failed with a syntax error, you would see the error description in this log, exactly as it would be described if the script were run from the command-line.

Auto-Rotating the Apache Logs

Some versions of Linux *auto-rotate* the Apache log files on a daily basis. This means that the entries stored in `/var/log/apache/access.log` and `/var/log/apache/error.log` may only describe today's activity. If this is the case, yesterday's activity is available in `access.log.1` and `error.log.1`, activity from the day before is in `access.log.2` and `error.log.2`, and so on. Most of the time, this

type of log rotation has a built-in upper limit, usually seven days. After the seventh day, the log file is deleted.

Log deletion is usually a good idea, as it prevents the log directory from taking up too much space on the Web server's hard drive. Often you can configure this by editing the file `/etc/logrotate.d/apache2` on Debian or Ubuntu, or `/etc/logrotate.d/httpd` on Red Hat.

Read the Apache Logs

You can access the Apache log files using any program that can open text files. On Windows this could be Notepad, or even using the `type` command at the DOS prompt. Opening the log file as a text file is okay for a server with very little activity, but it is not practical to view real-time data.

The `tail` utility is very useful for examining the access and error log files for real-time activity. It is available by default on Unix systems but you may need to manually install it on Windows. The log directory Apache stores its logs into differs, depending on your operating system. On Windows, this is `Apache Install Dir\logs`, on Debian and Ubuntu this is `/var/log/apache2/`, and on Red Hat this is `/var/log/httpd/`.

Extra

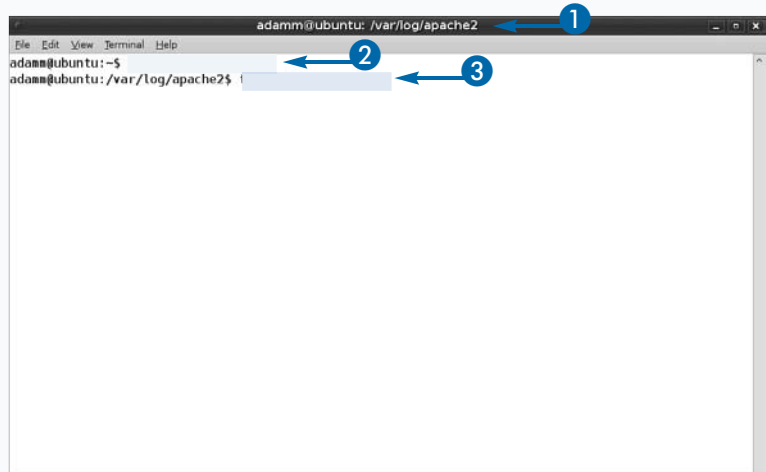
Tail can accept multiple files on the command line, making it possible to monitor both the `access.log` and `error.log` files at the same time. Tail on Windows is only a standard component on Windows Server 2003 and 2008 installations. Unfortunately, it is not standard on Windows desktop installations, such as XP, Vista, and 7.

A graphical version of Tail is available for Windows desktops at <http://tailforwin32.sourceforge.net>. Once downloaded, simply double-click the enclosed `tail.exe` file and open the `access.log` file.

Read the Apache Logs

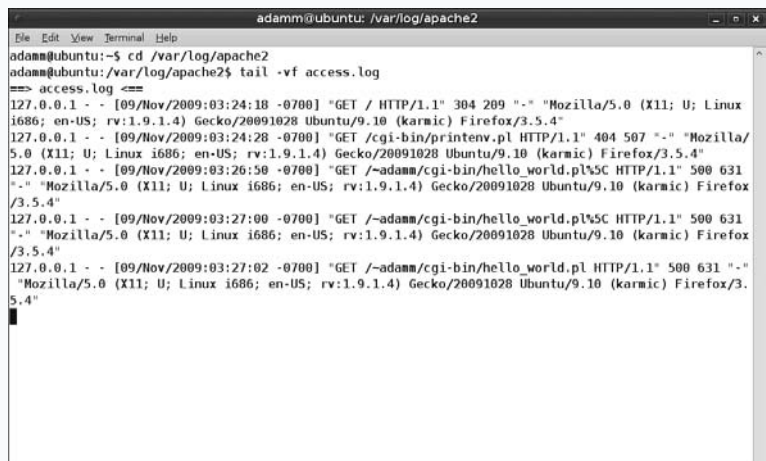
- 1 Open a command-line prompt.
- 2 Navigate to the log file directory.
- 3 Type `tail -vf access.log` (or `access_log`) and press Enter.

Note: You may need elevated privileges to view the log file. Use `sudo` or `su`.



A terminal window titled 'adamm@ubuntu: /var/log/apache2'. The prompt is 'adamm@ubuntu:~\$'. The user enters 'cd /var/log/apache2' and the prompt changes to 'adamm@ubuntu:/var/log/apache2\$'. The user then enters 'tail -vf access.log' and the prompt changes to 'adamm@ubuntu:/var/log/apache2\$ tail -vf access.log'. Three numbered arrows point to the terminal: arrow 1 points to the terminal title bar, arrow 2 points to the 'cd' command, and arrow 3 points to the 'tail' command.

The terminal displays the most recent log activity.



A terminal window titled 'adamm@ubuntu: /var/log/apache2'. The prompt is 'adamm@ubuntu:~\$'. The user enters 'cd /var/log/apache2' and the prompt changes to 'adamm@ubuntu:/var/log/apache2\$'. The user then enters 'tail -vf access.log' and the prompt changes to 'adamm@ubuntu:/var/log/apache2\$ tail -vf access.log'. The terminal displays the output of the tail command, showing the most recent log activity. The output is as follows:

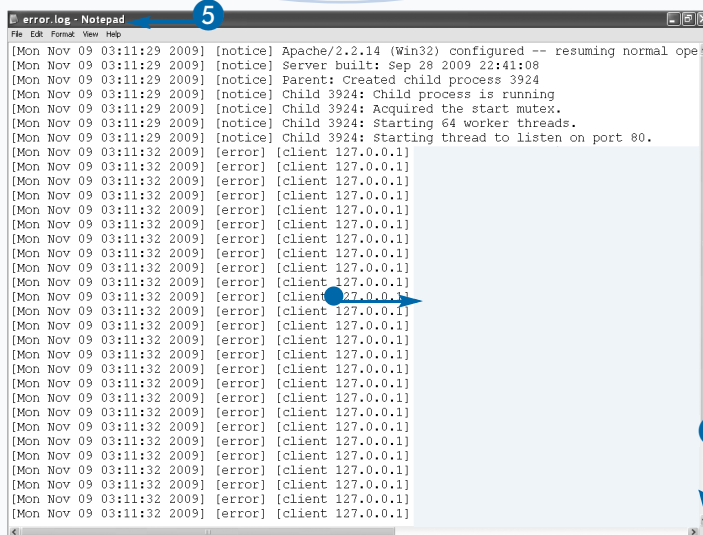
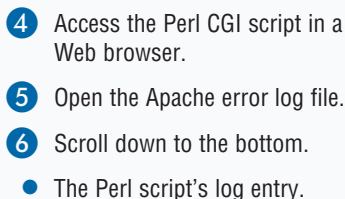
```
== access.log ==
127.0.0.1 - - [09/Nov/2009:03:24:18 -0700] "GET / HTTP/1.1" 304 209 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.4) Gecko/20091028 Ubuntu/9.10 (karmic) Firefox/3.5.4"
127.0.0.1 - - [09/Nov/2009:03:24:28 -0700] "GET /cgi-bin/printenv.pl HTTP/1.1" 404 507 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.4) Gecko/20091028 Ubuntu/9.10 (karmic) Firefox/3.5.4"
127.0.0.1 - - [09/Nov/2009:03:26:50 -0700] "GET /~adamm/cgi-bin/hello_world.pl HTTP/1.1" 500 631 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.4) Gecko/20091028 Ubuntu/9.10 (karmic) Firefox/3.5.4"
127.0.0.1 - - [09/Nov/2009:03:27:00 -0700] "GET /~adamm/cgi-bin/hello_world.pl HTTP/1.1" 500 631 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.4) Gecko/20091028 Ubuntu/9.10 (karmic) Firefox/3.5.4"
127.0.0.1 - - [09/Nov/2009:03:27:02 -0700] "GET /~adamm/cgi-bin/hello_world.pl HTTP/1.1" 500 631 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.4) Gecko/20091028 Ubuntu/9.10 (karmic) Firefox/3.5.4"
```


This can be especially useful if you are debugging a problem on a live Web site, but you cannot actually

Apply It

Try replacing `print STDERR` with the `warn` or `die` functions. Along with printing to standard-error, `warn` and `die` also append the script's name and line number into the error log, but only if the message does not end in a carriage-return (`\n`). The function `die` differs from `warn` in that it stops the execution of the script after relaying the message text. Alternatively, the Apache error messages can be forwarded back to CGI and the browser. For more information, see Chapter 12.

- 1 Open a Perl CGI script in a text editor.
- 2 Type **print STDERR TEXT;** and press Enter.
- 3 Save the Perl script.



Create an HTML Form

An HTML form collects information from a user by way of his browser, and relays that information to a Web server. Along with standard HTML formatting, an HTML form's syntax defines the input fields that can be populated, any predefined values, the buttons to submit the form, and the URL of the script accepting the form's data. To make this form useful, you need a CGI component at the other end of that URL which accepts the submitted data, interprets the data, reports any errors, and moves the user onto the next page according to the site's workflow.

An HTML form always begins with a form tag. This instructs the browser that an HTML form is beginning, and tells it the method type, and the URL that will receive the data when it is submitted.

```
<form method=TYPE action=URL>
```

The input fields within the form can include text boxes, pull-down lists, multi-select lists, check boxes, and radio buttons. You assign each of these input fields a name, which is an optional value within the input HTML tag. If you provide a value, it appears as the default value when the form is rendered; if you do not provide a value, the input field appears blank.

```
<input type=TYPE name=NAME value=VALUE>
```

The Submit button normally appears at the bottom of the form. It does not require a name or a value, but these fields are submitted to the CGI if you define them within the HTML form.

```
<input type=submit name=NAME value=VALUE>
```

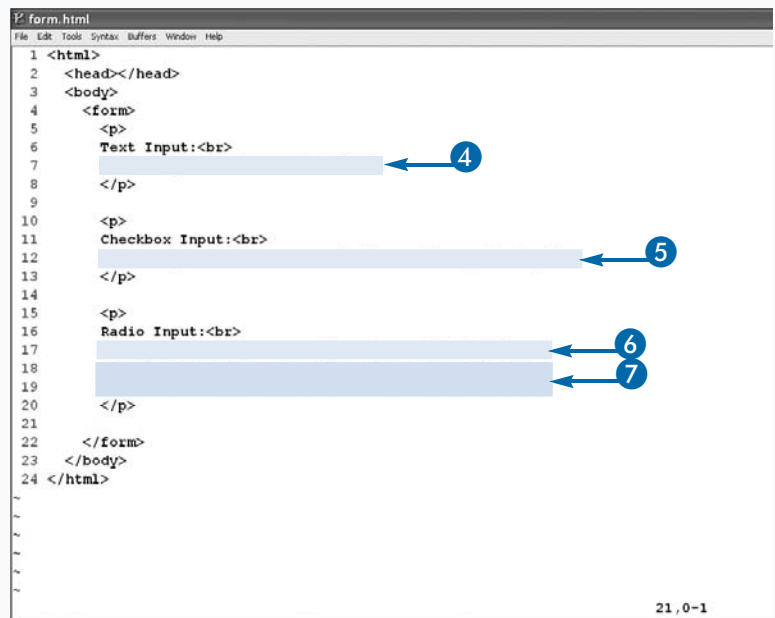
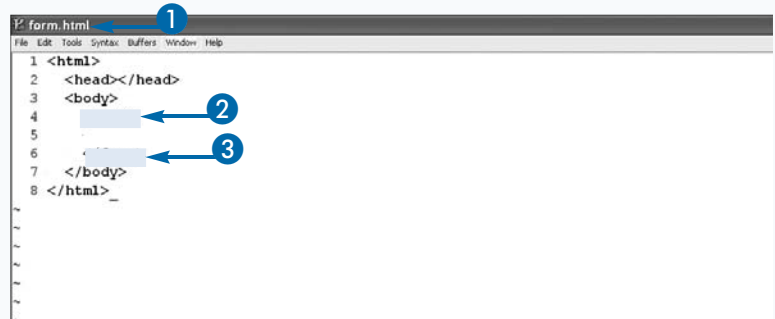
Finally, you complete the HTML form by closing the form tag.

```
</form>
```

Create an HTML Form

- 1 Open a static HTML script in a text editor.
- 2 Type `<form>` to start the HTML form.
- 3 Type `</form>` to close the HTML form.
- 4 Type `<input type=text name=field>` to create a text box.
- 5 Type `<input type=checkbox name=field value=value>text` to create a new check box and its text value.
- 6 Type `<input type=radio name=field value=value>text` to create a new radio button and its text value.
- 7 Duplicate the radio input fields, changing only the *value* and *text*.

Note: You can use radio buttons for multiple-choice questions, where the user can only select one choice at a time.



- 8 Type `<input type=submit>` to create a Submit button.

Note: If a Submit button contains a name or value field, this information is relayed to the CGI script as a standard key/value pair. As a result, you can define multiple buttons, each with a unique name and value. This helps the CGI script identify what the user clicked.

Open the HTML form in a browser.

- The input text field.
- The check box field.
- The radio button fields.
- The Submit button.

```

1 <html>
2 <head></head>
3 <body>
4   <form>
5     <p>
6       Text Input:<br>
7       <input type=text name=textfield>
8     </p>
9
10    <p>
11      Checkbox Input:<br>
12      <input type=checkbox name=checkbox value=option>option
13    </p>
14
15    <p>
16      Radio Input:<br>
17      <input type=radio name=radio value=option1>option 1
18      <input type=radio name=radio value=option2>option 2
19      <input type=radio name=radio value=option3>option 3
20    </p>
21
22    <input type=submit value=Submit Query>
23  </form>
24 </body>
25 </html>

```



Extra

An HTML form can be a static HTML file stored on a Web server, or it can be generated dynamically by another server-side CGI script.

To validate the fields submitted by the user, they should be *error corrected* by the accepting CGI script. This refers to the portion of the code that validates that an “e-mail address is an e-mail address” and a “phone number is a phone number.” Finally, if any errors are discovered, you need to communicate this back to the user.

A good practice is to re-display the same form, pre-populated with the user’s known-good values, and displaying a red background beside the fields where an error is discovered. This way, the user only needs to correct the missing fields and press the Submit button again.

It is also possible to use JavaScript to validate the user’s data as a *client-side control*. This can be used prior to the form’s actual submission as a convenience to the user. However, if used, client-side validation should only complement server-side validation, and you should not use it as the primary means of error correction. Clever users can bypass JavaScript validation controls; users cannot bypass Perl’s validation controls when you write them correctly.

Read HTTP GET/POST Parameters

When a user fills in an HTML form and submits it, a CGI script needs to receive the data, parse it, decode it, and act upon it. For example, the raw form data is delivered to the CGI script in this format:

```
firstname=John&lastname=Smith&email=jsmith%40domain.com
```

The parsing process involves splitting the string at each ampersand (&) to get each field, and then splitting each field at each equal sign (=) to get each key/value pair. The decoding process converts the non-alphanumeric characters from a percent-hexadecimal-ASCII representation, "%40", into the original character, "@".

HTML forms are submitted using a method, either GET or POST. The GET method provides the form data within the query string in the script's URL. The POST method does

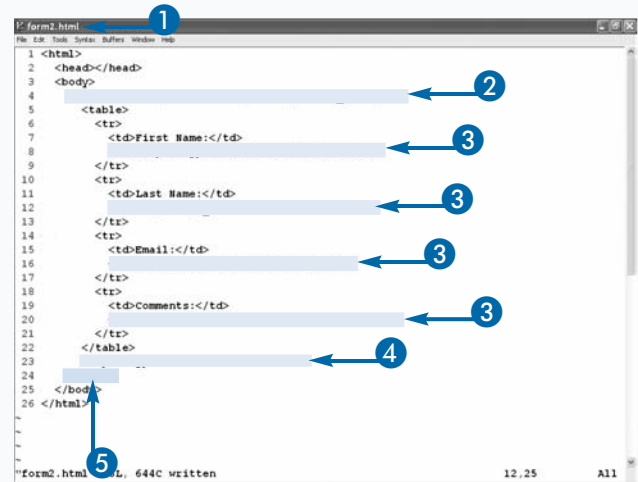
the same thing as GET, except the data is sent as a part of the HTTP request. The Perl script interprets this data as being a part of *standard input*, or STDIN. Regardless of whether you use GET or POST, the CGI Perl script must convert the raw data into a hash reference.

```
$param->{ 'firstname' }
$param->{ 'lastname' }
$param->{ 'email' }
```

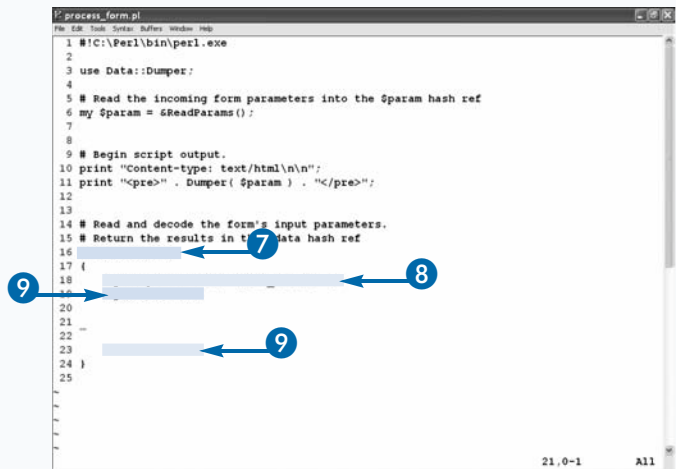
It is up to you to decide which method to use in your Web site. Generally speaking, GET forms are easier to test and debug new code on, as the data being submitted is visible right in the URL. POST forms are cleaner, but more difficult to validate visually without adding some additional Perl debug code. You should limit the total length of a URL to 256 characters; however, some browsers support more. Do not use GET if you are expecting a lot of data from the HTML form.

Read HTTP GET/POST Parameters

- 1 Open a static HTML script in a text editor.
 - 2 Type `<form method=get action="cgi-bin/script.pl">` to start the form.
 - 3 Insert a series of Input fields.
 - 4 Type `<input type=submit value="Submit">`.
- Note:** A submit input field uses the "value" attribute as the literal button text.
- 5 Type `</form>` to close the form.
- Save the file as `form.html`.



- 6 Open a new Perl CGI script.
- 7 Create a new ReadParams subroutine.
- 8 Type `my $input = $ENV{ 'QUERY_STRING' };` to declare a new scalar, and store the QUERY_STRING environment value.
- 9 Type `my $data = {};` to declare a new hash reference.



- 10 Type `return $data;` to return it.
- 11 Start a new `foreach` loop; split the input by the “&” character.
- 12 Split each element by the “=” character into key and value scalars.
- 13 Type `$val =~ s/%(..)/chr(hex($1))/ge;` to decode any encoded characters in `$val`.

```

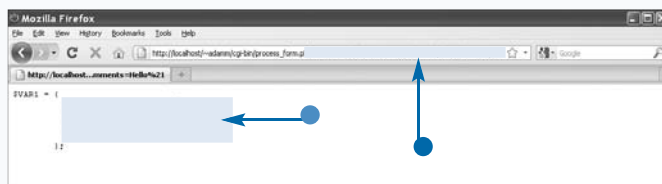
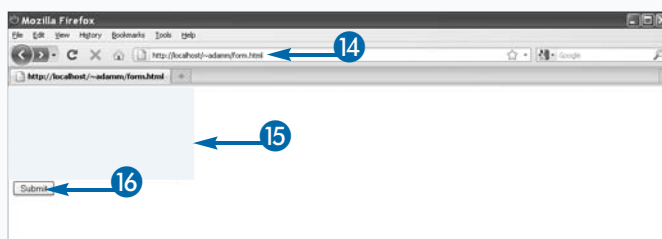
1 #!C:\Perl\bin\perl.exe
2
3 use Data::Dumper;
4
5 # Read the incoming form parameters into the $param hash ref
6 my $param = $ReadParams();
7
8 # Begin script output.
9 print "Content-type: text/html\n\n";
10 print "<pre>" . Dumper( $param ) . "</pre>";
11
12
13 # Read and decode the form's input parameters.
14 # Return the results to the $data hash ref
15 sub ReadParams
16 {
17     my $input = $ENV{ 'QUERY_STRING' };
18     my $data = {};
19     foreach ( split( $input, "&" ) ) {
20         my ( $key, $val ) = split( "=", $_ );
21         $val =~ s/%(..)/chr( hex( $1 ) )/ge;
22     }
23     return $data;
24 }
25
26
27 return $data;
28
29

```

- 14 Store each key and value into the hash reference.
- 15 Open the static HTML in a browser.
- 16 Type data into the fields.
- 17 Click Submit.

The CGI script executes.

- The submitted form data appears in the URL, encoded.
- The parsed form data returned by `Data::Dumper`.



Extra

To handle the `POST` method, you need to make two changes. First, the static HTML form needs to specify the `POST` method: `<form method=post action="cgi-bin/process_form.pl">`. Second, the script itself needs to retrieve data from *standard input*, or `STDIN`.

Insert this code prior to the first `foreach` loop on line 19.

Example:

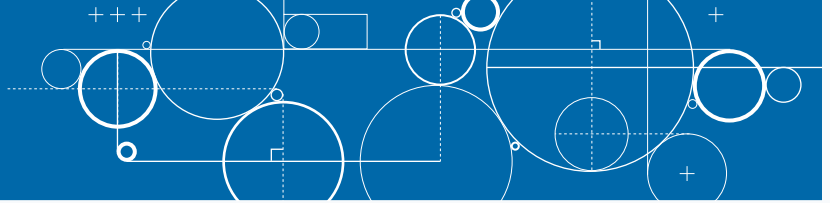
```

if ( $ENV{ 'REQUEST_METHOD' } eq 'GET' ) {
    $input = $ENV{ 'QUERY_STRING' };
}
else {
    while ( <STDIN> ) {
        $input .= $_;
    }
}

```

Once implemented, the difference between `GET` and `POST` to the end user is no data string in the URL after submitting the form. This is one example of parsing data through `GET` and `POST` methods. For a more robust solution, see Chapter 12.

Introducing Cookies



Web sites use cookies to store information on a user's browser. Each cookie contains a key/value pair, an expiry date, and a domain name, signifying who originally assigned the cookie. Browsers handle cookies using a security concept called the *same origin policy*. This means that data established by one site cannot be accessed by another site.

One common use for cookies is to track the user's session. When the user's browser opens a Web site for

the first time, the site assigns a uniquely identifying string to the browser (for example, `sessionId=74081257708423`). Each time the user opens another page on the Web site, the browser sends the same session cookie back to the server, as a form of identification.

CGI scripts in Perl need to assign and retrieve cookie data programmatically, so that the script can handle new and repeat users, and react to the user's particular session.

Using Cookies

Cookies travel within HTTP headers as an environment variable. The user's browser needs to have cookies enabled, and the CGI code needs to handle cookies.

Writing Cookies

Cookies are "written" to the user's browser by appending additional data into the HTTP *response* header, delivered prior to the HTML transfer. The browser parses the data and stores it within its internal configuration.

Reading Cookies

For each subsequent click on the Web site, all established cookies are sent back to the server in the HTTP *request* header. The CGI script then "reads" this information.

Expiring Cookies

Cookies may have a built-in expiry date configured by the Web site. Once a cookie expires, the browser no longer sends it back to the server in the HTTP *request* header.

Tracking Users with Cookies

You can track users with a *unique session identifier* that is assigned to their browser in the form of a cookie. The identifier does not store any personal information directly; instead, that data is cross-linked from a database on the server.

Securing Cookie Data

You can use cookies to store any type of data: personal information, personal preferences, or Web site history. The user's Web browser, not the Web server, stores and manages all cookie data. According to the cookie specifications, browsers are only allowed to relay cookie data back to the Internet domain that originally stored it. As a result, cookies stored by `abc.com` cannot be retrieved by `xyz.com`, and vice versa.

Unfortunately, an unauthorized Web site can employ an attack known as *cross-site scripting* to trick a browser into supplying it with data such as cookies. You can minimize this type of risk to your users by not storing sensitive data in the cookie, such as personal data, and by binding a cookie session key to a particular IP address.

Storing User-Specific Data

When developing cookies for your Web site, it is a good idea to gauge the impact on the user if the cookie were to be compromised. For example, storing a user's site preferences in a cookie is relatively safe. This may include a preferred language, time zone, and CSS layout. Storing this information is a convenience to the user; if it is lost, it can easily be reset.

You should store all other sensitive user data in databases hosted by your Web site, including name, address, and phone number, and you should not expose it to cookies directly.

Instead, a user-specific session key should be established on the browser when the user authenticates. This key can then be used to link to data in the database by the CGI.

Authorizing Users with Cookies

When a user accesses your Web site for the first time, a unique *temporary* session cookie should be assigned to the user's browser. This should happen prior to any authentication attempts by the user. Temporary cookies automatically expire when the user's browser is closed.

Authenticating the User

You should authenticate the user into your Web site through username and password validation over SSL. Once your Web site is satisfied with the user's response, a Perl CGI script can update your internal database and link the user's session cookie with her user profile.

It is up to you to decide if a user may authenticate from multiple browsers simultaneously. To do this, the user's profile in the database must be allowed to store multiple session cookies at the same time. Try to avoid this practice if you can. Forcing the user to re-validate on a new browser ensures that any earlier sessions, whether known by the user or not, are properly invalidated.

Tracking the User in the Session

For every click the user makes on your Web site, the browser provides the same session cookie back to the CGI. Your site needs to retrieve this session key and, based upon the user's activity, update the database with that key to process the activity.

For example, if the user adds an item to a shopping cart and purchases it, at least three unique CGI scripts are required. The first script lists the item with an "Add to Cart" link, the second script is the shopping cart summary, and the third script is the checkout process. Each CGI script needs to be aware of the user's session through the whole process. This includes not finding the item in the database, tracking multiple items in the cart, or a failed checkout process.

Several Perl modules exist that allow for tracking user sessions with a cookie. One popular option is the module `CGI::Session`, which is discussed in Chapter 12.

Logging-Off the User

The user may click a Log-Out button which executes CGI code that deletes the database link from the user's session cookie to her user profile. The user may also close her browser, thus killing the temporary session cookies.

If the user opens her browser again, she is forced to re-authenticate, which makes it look like she had properly "logged-off" earlier.

Expiring the Session

The user's session should only be allowed a small amount of inactivity time before the server automatically logs-off the user. Each click by the user within the context of a validated session should reset the internal database timestamp for the user, indicating the last time the user was active.

You can easily write an automatic program in Perl that constantly scans the user profile database. Users with an exceeded timestamp will have their session key deleted. If the user leaves his browser open for an extended period, no one can casually start clicking around later.

Deciding on the timeout threshold depends on the sensitivity of the data being stored. Some financial Web sites only allow five minutes of inactivity, while online forums allow for days of inactivity.

Store HTTP Cookies

You can use cookies to store information related to a particular user, browser, or session. When used to store an authentication token, a cookie can allow a user to “wander” throughout a Web site and maintain her session, regardless of page clicks, reloads, or sometimes even bookmarks.

A cookie is data stored as one or more key/value pairs, much like a Perl hash variable. Additional information for each pair can also be defined, such as an expiry date, a valid path, the domain name, and whether the cookie is intended only for SSL-encrypted connections.

When establishing a cookie, a CGI script sends the command `Set-Cookie` to the user’s browser through the HTTP *response* headers. The browser reads this data as it receives the Web page, and stores the cookie in its internal

database. The browser then sends the same data back to the Web server on each subsequent page load through the HTTP *request* headers. This allows the Web site to “remember” the user from one page click to the next.

Normally, users should never need to worry about cookies, or even that they exist. Cookies are merely a tool to help Web sites provide a more dynamic experience to the user. Responsible Web sites should only create a cookie when they need it, and set an appropriate expiry timeout.

Because cookies can store sensitive information, some Web browsers provide a way for the user to manually erase cookies from the computer. Firefox also provides a feature called *Private Browsing*. If the user enables Private Browsing, then all data collected by the browser is deleted when he disables Private Browsing. This includes all browsing history, cache, and cookies.

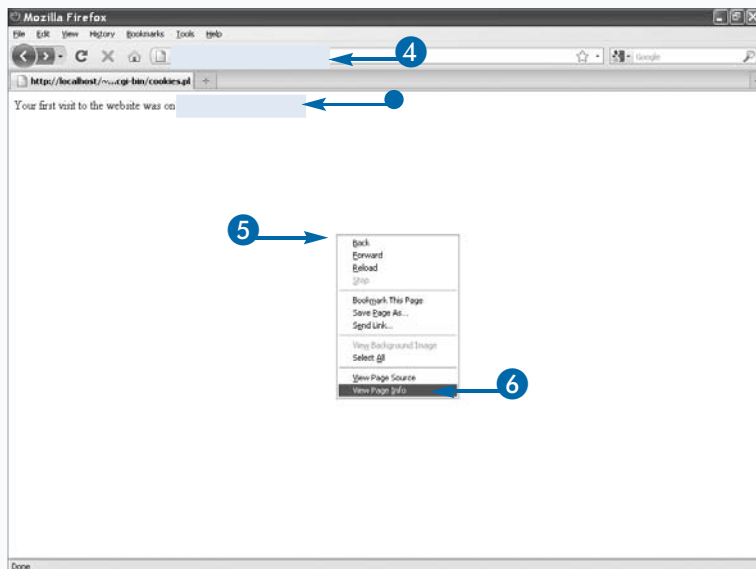
Store HTTP Cookies

- 1 Open a new Perl CGI script.
- 2 Type **print "Set-Cookie: key=value\n";** before the `Content-Type` header.
- 3 Display the cookie’s value in the CGI output to verify its value.

Save the CGI script.

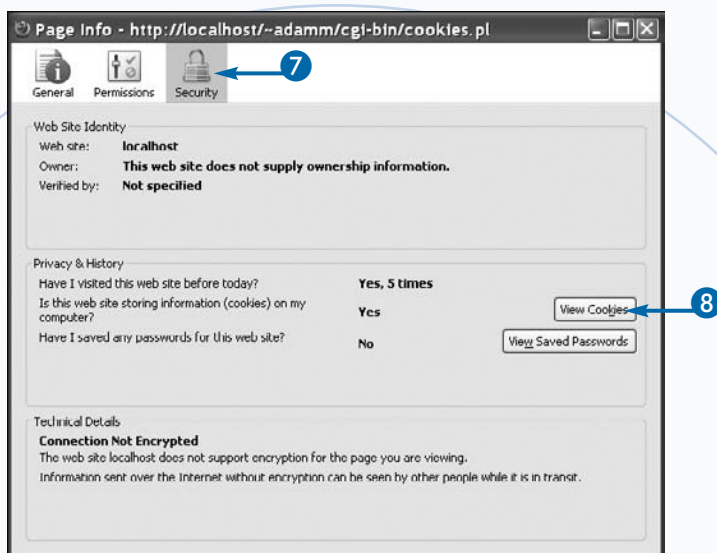
- 4 Open the CGI Perl script in a browser.
 - The browser displays the cookie’s value.
- 5 Right-click the Web page.
- 6 Click View Page Info.

```
1 #!C:\Perl\bin\perl.exe
2
3
4 my $time = localtime( time );
5
6
7 print "Content-type: text/html\n\n";
8
9 print "Your first visit to the website was on <b>" .
10
```



The Page Info dialog box appears, displaying the Security tab.

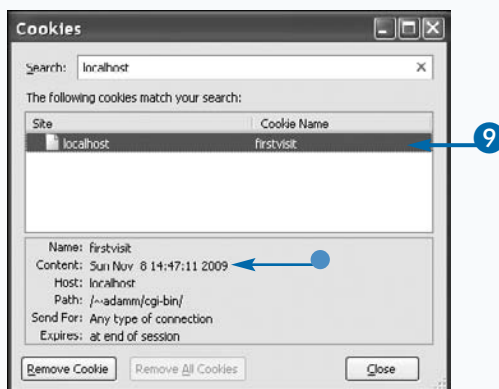
- 7 Click Security.
- 8 Click View Cookies.



The page's cookie summary window opens.

- 9 Select the cookie by its name.
 - The cookie's contents.

Note: Currently you are assigning the cookie a new value every time the page is loaded. You should not do this if you already have a value. For more information, see the section, "Retrieve HTTP Cookies."



Extra

The value stored with cookies must be *escaped* so it doesn't conflict with the "=" or ";" characters in the cookie formatted cookie string. This means changing individual characters into a multi-character value so that the original literal character cannot be misinterpreted. The easiest way to do this is to use regular expressions to replace all non-alphanumeric characters with their encoded values.

For an example of encoding and decoding non-alphanumeric cookies using regular expressions, see the section, "Retrieve HTTP Cookies."

Cookies without an expiry date are automatically considered temporary; they are deleted when the user closes her browser. Cookies with an expiry date are killed after the time elapses, even within the same browser session. Technically, if you leave a browser window running for several days, a temporary cookie could outlast a time-based cookie.

Cookies can also be written using other languages on the Web site, and then retrieved later within the Perl CGI, and vice versa. For example, if your site uses JavaScript, data can be written to the browser's cookie database using a JavaScript method, and retrieved using a Perl CGI script.

Retrieve HTTP Cookies

The ability to retrieve HTTP cookies back into a CGI script is as fundamental as writing them. By reading the data back in, your code receives the information it needs in order to fulfill your Web site's task online.

When a cookie is set, additional attributes such as the path, domain, and expiry date can be set. Unfortunately, the browser does not provide these extra attributes back to your script on follow-up page requests. You only receive the cookie keys and values as determined by the browser.

If a cookie you are expecting is not present, there could be multiple causes: the browser may have cookies disabled, the user could have deleted the cookie, or the cookie could have expired on its own. It is up to you to

decide how best to proceed. Naturally, testing the user's browser to ensure that cookies are enabled, and printing a very large warning message if they are not, is one of the best ways to forewarn users that your site uses cookies.

You can access the raw cookie data in Perl CGI through the `$ENV{ 'HTTP_COOKIE' }` variable. This environment variable follows a similar, but slightly modified structure, compared to the GET and POST submitted form strings:

`key1=value1; key2=value2; ...; keyN=valueN`

This data needs to be split apart and decoded into a hash reference, typically called `$cookies`. The process to do so is very close to reading HTTP GET and POST parameters that were described earlier in this chapter. From here, you can access the cookie value using `$cookies->{ KEY }`.

Retrieve HTTP Cookies

- 1 Open a CGI Perl script in a text editor.
- 2 Create a new `ReadCookies` subroutine.
- 3 Declare a new scalar variable, and store the `HTTP_COOKIE` environment variable.
- 4 Declare a new hash reference variable called `$data`, and return it.
- 5 Start a new `foreach` loop; split the input by the `';` characters.
- 6 Split each element by the `"="` character into key and value scalars.
- 7 Store each key and value into the hash reference.

```
1 #!C:\Perl\bin\perl.exe
2
3
4 my $time = localtime( time );
5
6 print "Set-Cookie: firstvisit=$time\n";
7 print "Content-type: text/html\n\n";
8
9 print "Your first visit to the website was on <b>" . $time . "</b>";
10
11
12
13 {
14
15
16
17
18
19
20 }
```


```
1 #!C:\Perl\bin\perl.exe
2
3
4 my $time = localtime( time );
5
6 print "Set-Cookie: firstvisit=$time\n";
7 print "Content-type: text/html\n\n";
8
9 print "Your first visit to the website was on <b>" . $time . "</b>";
10
11
12 sub ReadCookies
13 {
14     my $input = $ENV{ 'HTTP_COOKIE' };
15     my $data = ();
16
17     foreach (
18         my ( $key, $val ) =
19
20     )
21
22
23     return $data;
24 }
```

- 8 Type `my $cookies = ReadCookies();`.
- 9 Start a new conditional block. Test if `$cookies->{ key }` exists.
- 10 Type `$value = $cookies->{ key };` to retrieve its value if it does exist.
- 11 Set the cookie if it does not exist.
- 12 Save the CGI script.

```

1 #!C:\Perl\bin\perl.exe
2
3
4 my $time = localtime( time );
5
6
7 if (
8
9
10
11
12
13 print "Content-type: text/html\n\n";
14
15 print "Your first visit to the website was on <b>" . $time . "</b>";
16
17
18 sub ReadCookies
19 {
20     my $input = $ENV{ 'HTTP_COOKIE' };
21     my $data = {};
22
23     foreach ( split( ' ', $input ) ) {
24         my ( $key, $val ) = split( "=", $_ );
25
26         $data->{ $key } = $val;
27     }
28
29     return $data;
30 }

```

- 13 Open the CGI Perl script in a browser.
 - The browser displays the cookie's value.
- 14 Click  to reload the Web page.

The displayed value does not change.



Extra

Restart your Web browser and try the CGI script again. You should now see a new timestamp, and subsequent reloads should keep that time. Remember, closing your browser deletes the temporary cookie, and so the Web site will think you are a new user when you later visit the Web site.

The data being stored and retrieved in the cookie is not yet encoded properly. If the key or value contained the characters “=” or “;” it would conflict with HTTP_COOKIE’s syntax. All non-alphanumeric text in a cookie’s value should be encoded into its percent-hexadecimal-ASCII value. You could apply this encoding statement just before line 11:

Example:

```
$time =~ s/(\\W)/sprintf( "%02x", ord $1 )/ge;
```

You also need the decoding counterpart statement. This time, before line 26, add the following line:

Example:

```
$val =~ s/%(\\w\\w)/chr( hex( $1 ) )/ge;
```

These two statements may seem confusing, especially if you are new to regular expressions. While they do convert non-alphanumeric strings to their hexadecimal counterpart and back, it is important to understand why you need them. Chapter 12 introduces the Perl CGI Library, which makes reading and writing cookies easier in Perl.

Send an E-Mail Message

Sending an e-mail message from a CGI script is a very useful feature to add to your Web site. You can use it to validate an e-mail address entered by a new user, notify a user when his product has been shipped, or alert an administrator if an unexpected problem occurs. Perl does not have a built-in e-mail function, but it does have several support modules that can communicate to a variety of mail servers. One popular module available on CPAN is called Email::Sender.

To send an e-mail message, you need access to a *Mail Transport Agent*, or MTA. The MTA is a computer program or service that understands how e-mail messages are routed out to the Internet. On most Unix systems, you can use the *sendmail* program as a local MTA; otherwise, you need to use an Internet Service Provider's SMTP server as a remote MTA.

The function `sendmail` is automatically exported by `Email::Sender::Simple`. It triggers the actual process of sending the e-mail message, and automatically attempts to identify your MTA.

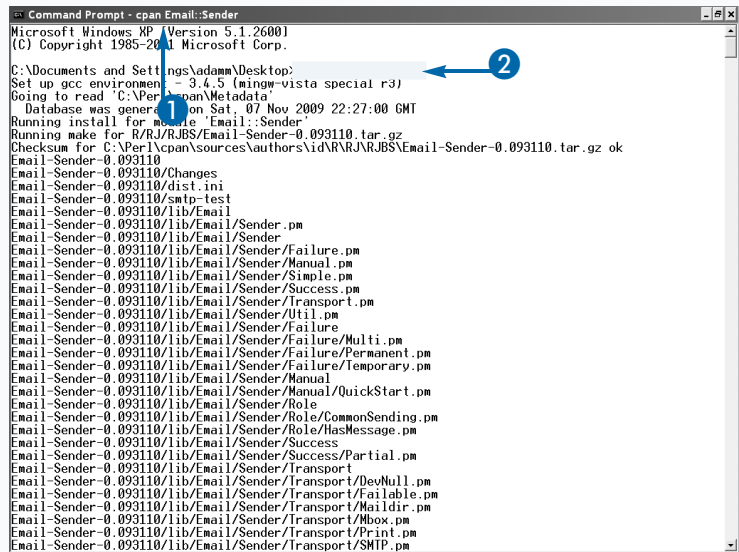
Normally, if `sendmail` fails to send the message, the script is killed and an error message is printed to the screen. When a CGI script dies prematurely, the entire output to the Web browser changes to reflect a generic server-side error. Therefore, you need to trap the exception of `sendmail` with `eval`, display an appropriate error message, and let the program continue.

When sending e-mail messages, make sure the “from” address is valid! Some MTAs check this before accepting an outgoing message. Also, if you are using SMTP, verify if your provider requires SSL encryption.

Send an E-Mail Message

- 1 Open a Terminal window.
- 2 Type **`cpan Email::Sender`** and press Enter.

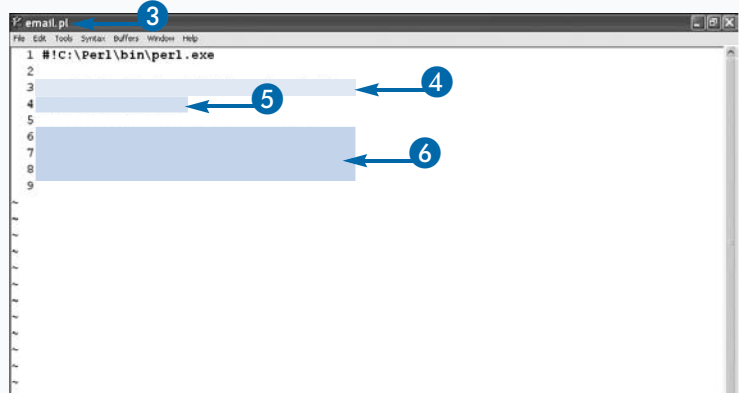
The Email::Sender module installs.



```
Command Prompt - cpan Email::Sender
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\adam\Desktop>
Set up gcc environment: - 3.4.5 (mingw-vista special r3)
Going to read 'C:\Perl\cpan\Metadata
Database was generated on Sat, 07 Nov 2009 22:27:00 GMT
Running install for module 'Email::Sender'
Running make for R/RJ/RJBS/Email-Sender-0.093110.tar.gz
Checksum for C:\Perl\cpan\sources\authors\id\R\RJ/RJBS/Email-Sender-0.093110.tar.gz ok
Email-Sender-0.093110
Email-Sender-0.093110/Changes
Email-Sender-0.093110/dist.ini
Email-Sender-0.093110/Makefile
Email-Sender-0.093110/lib/Email
Email-Sender-0.093110/lib/Email/Sender.pm
Email-Sender-0.093110/lib/Email/Sender
Email-Sender-0.093110/lib/Email/Sender/Failure.pm
Email-Sender-0.093110/lib/Email/Sender/Manual.pm
Email-Sender-0.093110/lib/Email/Sender/Simple.pm
Email-Sender-0.093110/lib/Email/Sender/Success.pm
Email-Sender-0.093110/lib/Email/Sender/Transport.pm
Email-Sender-0.093110/lib/Email/Sender/Util.pm
Email-Sender-0.093110/lib/Email/Sender/Failure
Email-Sender-0.093110/lib/Email/Sender/Failure/Multi.pm
Email-Sender-0.093110/lib/Email/Sender/Failure/Permanent.pm
Email-Sender-0.093110/lib/Email/Sender/Failure/Temporary.pm
Email-Sender-0.093110/lib/Email/Sender/Manual
Email-Sender-0.093110/lib/Email/Sender/Manual/QuickStart.pm
Email-Sender-0.093110/lib/Email/Sender/Role
Email-Sender-0.093110/lib/Email/Sender/Role/CommonSending.pm
Email-Sender-0.093110/lib/Email/Sender/Role/HasMessage.pm
Email-Sender-0.093110/lib/Email/Sender/Success
Email-Sender-0.093110/lib/Email/Sender/Success/Partial.pm
Email-Sender-0.093110/lib/Email/Sender/Transport
Email-Sender-0.093110/lib/Email/Sender/Transport/DevNull.pm
Email-Sender-0.093110/lib/Email/Sender/Transport/Failable.pm
Email-Sender-0.093110/lib/Email/Sender/Transport/Maildir.pm
Email-Sender-0.093110/lib/Email/Sender/Transport/Mbox.pm
Email-Sender-0.093110/lib/Email/Sender/Transport/Print.pm
Email-Sender-0.093110/lib/Email/Sender/Transport/SMTP.pm
```

- 3 Open a Perl script in a text editor.
- 4 Type **`use Email::Sender::Simple qw(sendmail);`** to import the `Email::Sender::Simple` module.
- 5 Type **`use Email::Simple;`** to import the `Email::Simple` module.
- 6 Type **`my $email = Email::Simple->create(`**
`);` to initialize the module.



```
email.pl
#!/usr/bin/perl
use Email::Sender::Simple qw(sendmail);
use Email::Simple;

my $email = Email::Simple->create(
);
```


7 Type **header => []**, to start a new array reference.

8 Insert the **to**, **from**, and **subject** fields and their values.

Note: Use single-quotes with the e-mail addresses! Double-quotes will expand the @, which Perl will interpret as a literal array. If you must use double-quotes, escape the at-sign: \@.

9 Type **body => MESSAGE** for the e-mail message body.

10 Type **eval {** to start a new evaluation code block.

11 Type **sendmail(\$email);**

12 Type **}** or **do {** to catch the failure.

13 Print a failure message, using `$@->message`.

14 Type **}** to close the catch block.

Save the script and run the program.

```

1 #!C:\Perl\bin\perl.exe
2
3 use Email::Sender::Simple qw(sendmail);
4 use Email::Simple;
5
6 my $email = Email::Simple->create(
7     [
8         to => 'jsmith@domain.com',
9         from => 'support@website.net',
10        subject => 'Test email',
11    ],
12    body => <<EOF,
13    Hello,
14    This is a test of the email CGI program. Please do not reply.
15    Thank you!
16    EOF
17 );
18
19 # Prevent sendmail() from killing the script with eval. We must "try" the
20 # command in the first code block. If it dies, "catch" the failure in the
21 # second code block.
22 eval {
23     sendmail($email);
24 }
25
26 if ($?) {
27     print "Email sent successfully!\n";
28 } else {
29     print "Email failed: " . $@->message . "\n";
30 }

```

```

31
32 #email.pl 30L, 652C written
33
34 30,0-1 All

```

Extra

By default, the module tries to identify the most appropriate local MTA, and a default SMTP server. If both attempts fail, your code may need to specify an SMTP server for transport. At the top of the script, import the SMTP transport module:

```
use Email::Sender::Transport::SMTP;
```

Next, define the transport handle:

```
my $smtp = { Email::Sender::Transport::SMTP->new( {
    host => 'smtp.myisp.com'
} )};
```

Finally, add the SMTP transport to the `sendmail` command:

```
sendmail($email, { transport => $smtp } );
```

The SMTP transport module supports SSL authentication, a custom port number, and other attributes. For more information, see the module's documentation:

```
perldoc Email::Sender::Transport::SMTP
```

If you simply want to see the message printed to the screen, use `Email::Sender::Transport::Print` instead. To entirely disable sending e-mail, but still have the program return success, use `Email::Sender::Transport::DevNull`.

Introducing the Built-In CGI Library

Perl has a built-in CGI library that has been a standard part of the Perl distribution since 1998. Countless Perl CGI developers have used it to quickly develop CGI scripts to handle mundane tasks such as parsing parameters, reading and writing cookies, and generating HTML tags and forms.

Even though the module is still under active development, many developers consider the CGI library to be outdated in the way it handles some tasks. While there are now better modules available that you can use to generate

static HTML tags and forms, the real strength of the CGI library is in its ability to translate raw incoming CGI data into Perl variables and functions.

The CGI library has shipped with every Perl distribution since version 5.0, but there may be a more recent version available on CPAN. You can upgrade to the latest release on the command-line with `cpan CGI`.

All of the CGI library documentation is available under the Perl Documentation program. You can access it on the command-line with `perldoc CGI`.

Use the CGI Library

The CGI library exports several functions to Perl. You can access these functions as methods through a CGI object handle, or you can export them as standard routines.

Create CGI Objects to Access Methods

As described in Chapter 8, you can access the CGI library through a module reference scalar. Through this reference, you can access the module's subroutines as methods.

The following syntax is recommended for larger sites, and when you specifically need an object-orientated interface to handle the CGI library.

```
use CGI;
my $cgi = new CGI;
$cgi->method;
```

Import Methods as Functions

An alternative to using a reference scalar is to import the methods directly as functions:

```
use CGI methods;
function;
```

The methods that are imported depend on what you specify on the first line. You can cite the literal function names, or a predefined function category. For example, if you are only interested in the CGI-handling methods, type **use CGI 'cgi'**; to gain access to functions such as `param`, `cookie`, `header`, and so on.

For more information, see “Using the Function-Oriented Interface” in the PerlDoc CGI documentation.

Available Methods

Regardless of what syntax you use to access the CGI library, the functionality is the same. There are over two hundred methods available, but the majority are simple shortcuts for writing raw HTML. There are methods that handle complex CGI tasks by auto-adjusting themselves according to your Web browser and server software.

CGI Routines

The three most useful CGI-related methods available in the CGI library are the `header`, `param`, and `cookie` routines. These provide an easy interface to accessing the most common functionality required by any CGI Perl Script.

Other available CGI routines include shortcuts for accessing the various HTTP environment variables, accepting uploaded files, and identifying authenticated user sessions.

One advantage of the CGI library's built-in CGI routines is that they strive to be *handler-independent*. This means that they work regardless of whether you use the standard Apache CGI handler, `mod_perl`, or FastCGI. If you try to code this functionality manually, you may require a separate procedure for each handler type.

HTML Routines

Most HTML routines provided by the CGI library are rather antiquated. You can use them for quickly developing technically correct HTML syntax, but they do not necessarily save any effort. If you are already familiar with HTML syntax, use the `HTML::Template` module described in Chapter 13. You are not saving yourself any effort by using the CGI library's HTML routines. They are more tedious than useful.

These routines are automatically available as methods when creating a CGI module reference, but you must specify `:html` as a module argument if you want to import the methods as functions.

Text Formatting

You can simply format text by using the HTML tag name as a method, the arguments of which are treated as the tag's content:

```
print h1( "Hello!" );
```

This is exactly the same as printing `<h1>Hello!</h1>`. Therefore, you can represent any `<tag>text</tag>` as `tag(text)`.

Tables

Tables follow a “tag-to-method” structure that is similar to text tags, but when the tags are nested, you can build the entire table in a single Perl statement:

```
print table(
    tr( th( [ undef, "Col1", "Col2", "Col3" ]
    ),
    tr( td( [ "Row1", "1:1", "1:2", "1:3" ] )
    ),
    tr( td( [ "Row2", "2:1", "2:2", "2:3" ] )
    ) );
```

While it is great that this is possible, you are not really saving much effort.

Forms and Input Fields

Forms generation also follows the same “tag-to-method” structure, but there are some new routines that provide some automation. Routines such as `textfield`, `checkbox`, `radio_group`, and `popup_menu` handle multiple HTML tags related to input, and accept parameters to populate their attributes.

Generating Dynamic Content

Producing a dynamic experience for the user is the whole point behind CGI. The CGI library provides three routines that simplify CGI functionality in Perl.

Headers

The `header` routine provides a shortcut to printing “Content-type: text/html”. It can take a series of arguments in different formats. When you use it with no arguments, it sends the default MIME type, “text/html”:

```
print header();
```

If you use exactly one argument, it is interpreted as the MIME type:

```
print header( "text/html" );
```

You can use parameters to define multiple headers, giving your Perl CGI script full control over the HTTP response:

```
print header( -type => "text/html", -expires =>
    "+1m",
    -cookie => $cookie );
```

Parameters

You use the `param` routine to retrieve HTML form fields that have been submitted to your CGI script. The function takes one argument, the field name, and returns its value if defined.

Cookies

You use the `cookie` routine to set and retrieve cookies from the browser. When you use it to set a cookie, it accepts an anonymous hash that defines the cookie's name, value, domain, path, and expiry date. The results of this function should be forwarded to `header` so that the cookie is included in the outgoing HTTP response header:

```
my $cookie = cookie( -name => "lang", -value =>
    "en_US",
    -expires => "+7d" );
print header( -cookie => $cookie );
```

When reading cookies, `cookie` accepts one argument: the cookie name. It returns the value of the cookie, if it is defined:

```
unless ( $lang = cookie( "lang" ) ) {
    $lang = "en_US"; # Set language to en_US
    if no cookie found. }
```

Import the CGI Library as an Object

You can access the CGI library's routines by initializing a module object reference after you import the CGI module. You can use this to follow a more object-oriented syntax when interacting with the CGI library, and to handle multiple copies of the CGI library that are loaded into memory. This process happens in three steps. First you import the CGI library module, then initialize the module into a new object reference variable, and finally use that variable to access the method:

```
use CGI;
my $cgi = new CGI;
$cgi->method();
```

When importing the module, you can provide additional arguments, *pragmas*, to change how the CGI library loads:

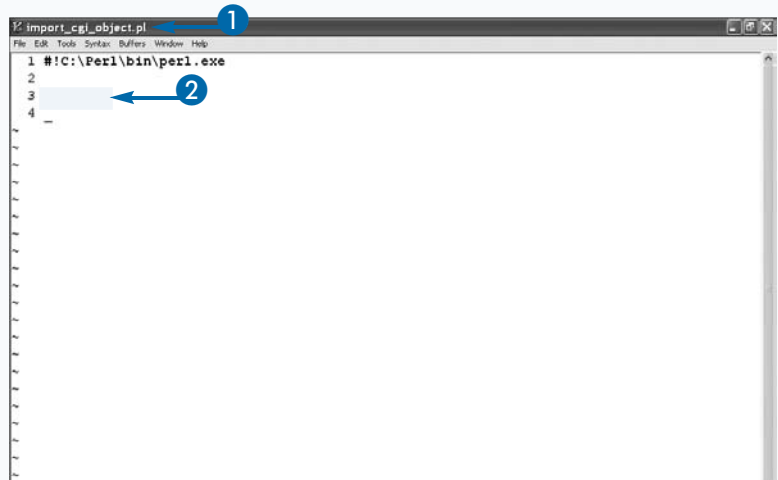
```
use CGI qw( ARG$... );
```

The most useful pragma is `-debug`. This allows you to simulate submitted form data on the command-line by manually typing the key/value pairs as arguments. The syntax of the pragma argument is fairly open. Many Perl developers prefer the `qw` function to provide arguments to modules in this fashion. Because the arguments are effectively arrays, `qw` converts a series of words into a list. To add in multiple arguments, just amend them within `qw`, like `use CGI qw(-debug :cgi :html);` — this appears cleaner than `use CGI '-debug', ':cgi', ':html';` though either method is equally valid.

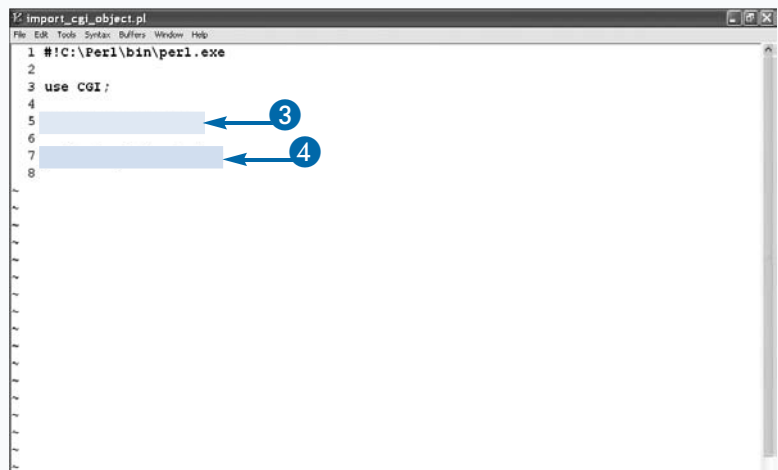
The CGI library may instill some overhead latency as it is imported into your program. If this latency becomes a problem for your project, you can mitigate this by enabling the FastCGI or `mod_perl` enhancements. The complete list of pragmas, along with all routines, syntax, and examples, is available in the CGI library's documentation.

Import the CGI Library as an Object

- 1 Open a Perl CGI script in a text editor.
- 2 Type **use CGI;** and press Enter.



- 3 Type **my \$cgi = new CGI;** to create an object reference.
- 4 Type **\$cgi->method();** to use the method through the object reference.



Note: Only the methods that need to send data directly to the browser require you to include the code, `print`.

Import the CGI Library's Routines as Functions

You can export the CGI library's routines as functions by providing one or more literal or group names as arguments when importing the library. You can use this technique to avoid typing the module reference every time you need a routine, resulting in cleaner code. This process happens in two steps. First, import the CGI library module with a series of arguments, specifying the literal routines or groups of routines. Second, wherever required, access these routines as normal functions.

```
use CGI qw( ARGUMENTS... );
function();
```

There are some predefined group names that automatically import common CGI library routines as functions for your program. The group `:cgi` imports

all CGI-related routines such as `header`, `cookie`, and `param`. The group `:html` imports HTML-related routines such as `h1`, `b`, `p`, and `table`. The group `:standard` imports the CGI, HTML, and routines specific to generating forms. The group `:all` imports every routine defined by the CGI library as a function.

You must also define additional pragma arguments as the literal function names. For example, if you only require the `header` and `param` functions but want to bypass everything else provided by the group `:cgi`, you can use:

```
use CGI qw( header param );
my $var = param( key );
print header();
```

The CGI library may instill some overhead latency when running your code. If this latency becomes a problem, you can mitigate this by enabling the FastCGI or `mod_perl` enhancements.

Import the CGI Library's Routines as Functions

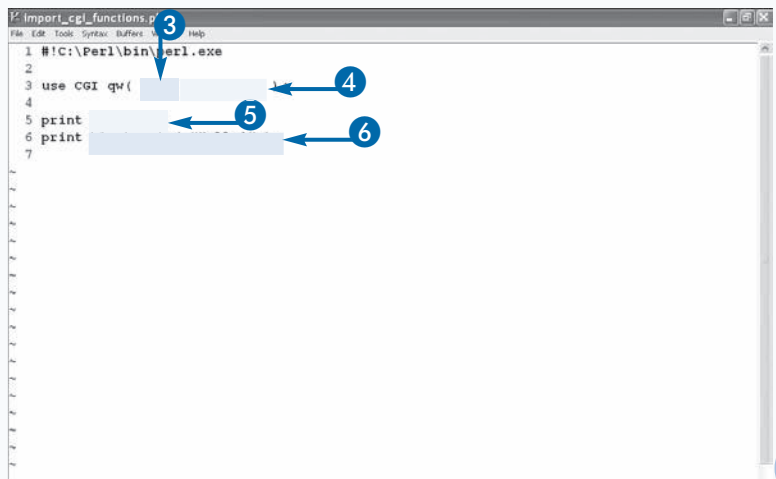
- 1 Open a Perl CGI script in a text editor.
- 2 Type `use CGI qw();` and press Enter.

Note: The `qw` function converts words into an array list.



- 3 Type the function group's name.
- 4 Type the function's name.
- 5 Type `function()` to use a method exported from the group.
- 6 Type `function()` to use a method explicitly exported.

Note: Only the methods that need to send data directly to the browser require you to include the code, `print`.



Read HTTP GET/POST Parameters with the CGI Library

The CGI library has a built-in routine called `param` that you can use to access fields submitted from HTML forms, using either the GET or POST method.

This saves you from having to manually parse `$ENV{'QUERY_STRING'}` or standard-input in your Perl script.

To read form data in, import the CGI library, and then call the `param` routine with the field name as an argument. Store the results into a scalar variable. If you want to use the routine as a function, specify the `:cgi` function group when loading the CGI library:

```
VAR = param( field );
```

When retrieving submitted HTML form data, it is possible for some input fields to have multiple values, as is usually the case with multiple-option check boxes. If this happens,

you must store your HTML parameter into an array variable, not a scalar.

If you use `param` without any arguments, it returns true if any parameters were submitted at all, and false if none were submitted. This can be used by conditional statements to validate if a form has been submitted yet or not. This routine is very robust in that it properly handles GET and POST submissions interchangeably. There is no need to change your code if you want to switch the form's method from one type to another. It even correctly parses forms using either `application/x-www-form-urlencoded` or `multipart/form-data` encoding.

If you are using any other non-standard encoding in your form, `param` does not automatically parse the incoming data feed. Instead, the only parameter name that it accepts is `POSTDATA`. This contains the raw, unprocessed data stream that is most often used for AJAX and XML processing.

Read HTTP GET/POST Parameters with the CGI Library

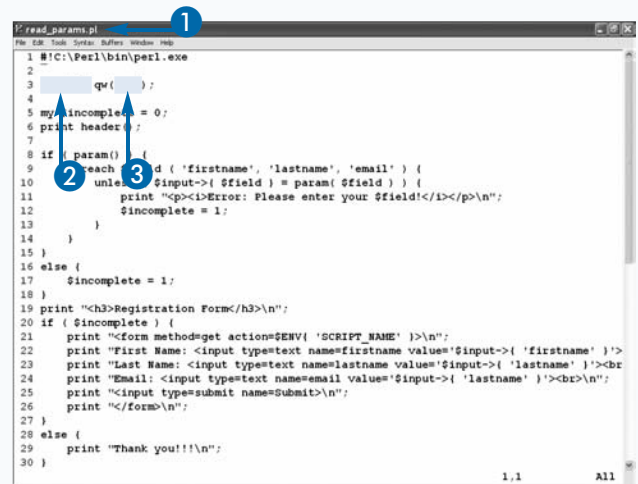
- 1 Open a Perl CGI script in a text editor.
- 2 Type `use CGI qw();` to import the CGI library module.
- 3 Type `:cgi` to import the 'CGI' group of functions.

Note: You can also use the alternative object-oriented method described in the section, "Import the CGI Library as an Object."

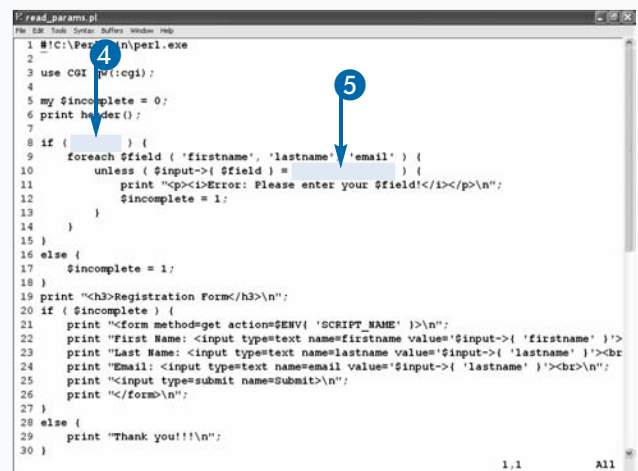
- 4 Use `param()` without arguments to test for any parameters.
- 5 Use `param(name)` to retrieve a parameter's value by its name.

Note: This example is storing the submitted data into the `$input` hashref. The `unless` code block will run only if `param($field)` is empty.

- 6 Save the Perl CGI script.



```
1 read_params.pl
2 #!C:\Perl\bin\perl.exe
3
4
5 my $incomplete = 0;
6 print header();
7
8 if ( param() ) {
9     foreach $field ( 'firstname', 'lastname', 'email' ) {
10         unless ( $input->{$field} = param( $field ) ) {
11             print "<p><i>Error: Please enter your $field!</i></p>\n";
12             $incomplete = 1;
13         }
14     }
15 }
16 else {
17     $incomplete = 1;
18 }
19 print "<h3>Registration Form</h3>\n";
20 if ( $incomplete ) {
21     print "<form method=get action=$ENV{'SCRIPT_NAME'}>\n";
22     print "First Name: <input type=text name=firstname value='$input->{ 'firstname' }>\n";
23     print "Last Name: <input type=text name=lastname value='$input->{ 'lastname' }>\n";
24     print "Email: <input type=text name=email value='$input->{ 'email' }>\n";
25     print "<input type=submit name=Submit>\n";
26     print "</form>\n";
27 }
28 else {
29     print "Thank you!!\n";
30 }
```



```
1 read_params.pl
2 #!C:\Perl\bin\perl.exe
3 use CGI qw( :cgi );
4
5 my $incomplete = 0;
6 print header();
7
8 if ( param() ) {
9     foreach $field ( 'firstname', 'lastname', 'email' ) {
10         unless ( $input->{$field} = param( $field ) ) {
11             print "<p><i>Error: Please enter your $field!</i></p>\n";
12             $incomplete = 1;
13         }
14     }
15 }
16 else {
17     $incomplete = 1;
18 }
19 print "<h3>Registration Form</h3>\n";
20 if ( $incomplete ) {
21     print "<form method=get action=$ENV{'SCRIPT_NAME'}>\n";
22     print "First Name: <input type=text name=firstname value='$input->{ 'firstname' }>\n";
23     print "Last Name: <input type=text name=lastname value='$input->{ 'lastname' }>\n";
24     print "Email: <input type=text name=email value='$input->{ 'email' }>\n";
25     print "<input type=submit name=Submit>\n";
26     print "</form>\n";
27 }
28 else {
29     print "Thank you!!\n";
30 }
```


- 7 Load the Perl CGI script in a Web browser.

The first `param()` test runs and returns false. The form you created appears, but with no additional text.

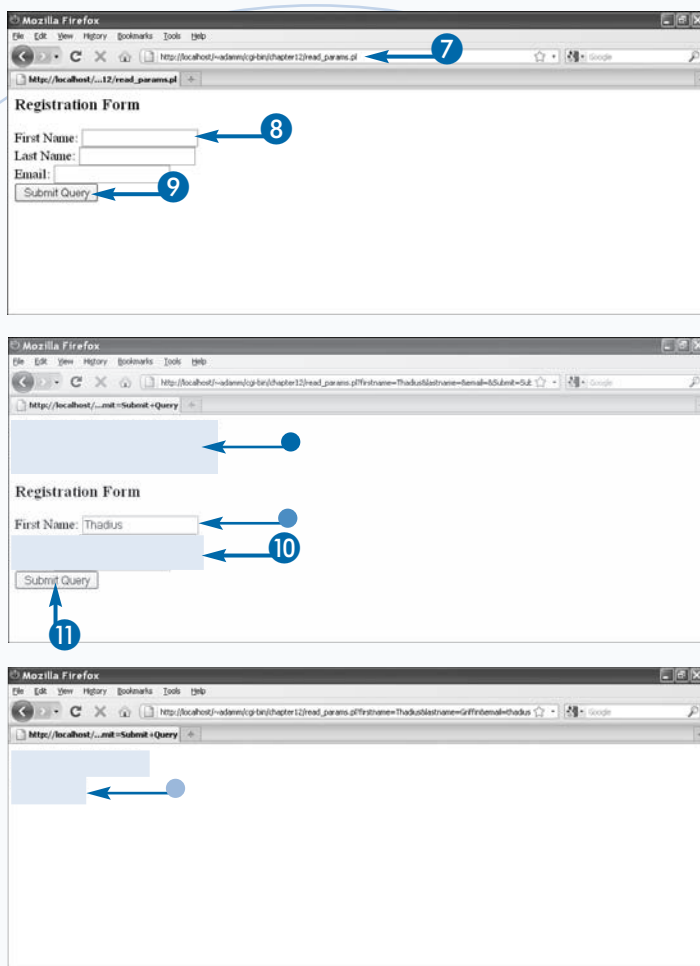
- 8 Populate the first input field only.
- 9 Click Submit Query.

The CGI script reloads.

- The script warns about the missing data.
 - The supplied field is pre-populated.
- 10 Populate all remaining fields.
 - 11 Click Submit Query.

The CGI script reloads.

- A message appears, telling you that the script has received all of the data.



Extra

It is possible to combine the HTML form and the parameter analysis into a single CGI script. You can do this by splitting your CGI code into two components: analysis and display. Before doing this, you must divide your program's *workflow* into at least three stages that the user will progress through: no data submitted, partial data submitted, and all data submitted.

To support these three stages, you need to organize the code logically using the two components. If the analysis component identifies that nothing has been submitted, the script displays a blank form. If the analysis identifies that partial data has been submitted, the script displays a message and the original form again. Finally, if the analysis is completely satisfied with the data, the script displays something entirely different.

If the user supplies only partial data, it is a good idea to reload the page and display the form again but pre-populate the correctly populated fields with the user's previous values. This helps save the user from having to re-type everything, which is especially beneficial if this is a large form with many fields. Naturally, some fields, such as password prompts and credit-card numbers, should never be pre-populated in forms.

Store HTTP Cookies with the CGI Library

You can use cookies to store information related to a particular user, browser, or session. You can use the CGI library to automatically provide the Set-Cookie syntax in the HTTP response header. This is conveyed by the `cookie` routine.

To store a cookie, import the CGI library, then call the `cookie` routine with the cookie's name and value as arguments. This returns an object handle that contains the cookie's definition. If you want to use the routine as a function, specify the `:cgi` function group when loading the CGI library:

```
VAR = cookie( name, value );
```

If you need to set additional cookie attributes, a second syntax format is supported by the routine. This allows you to define the cookie by the attributes using only the

keys that are required. You must prefix the attributes with a dash: `-name`, `-value`, `-expires`, `-domain`, `-path`, and `-secure`:

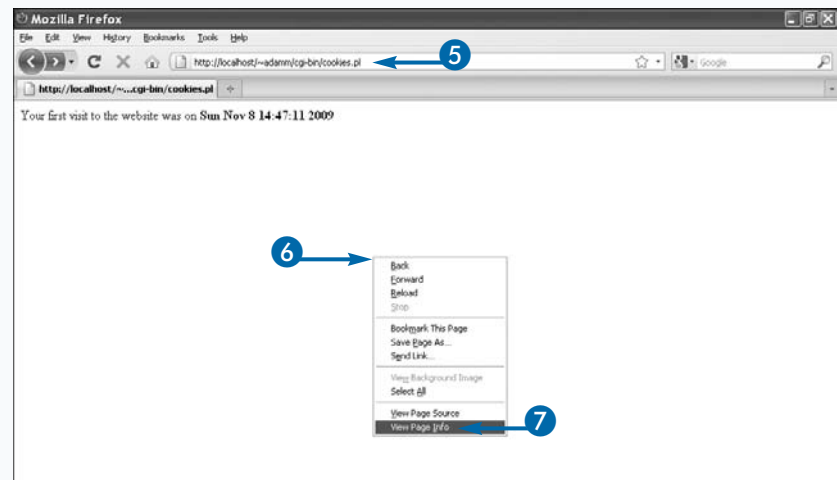
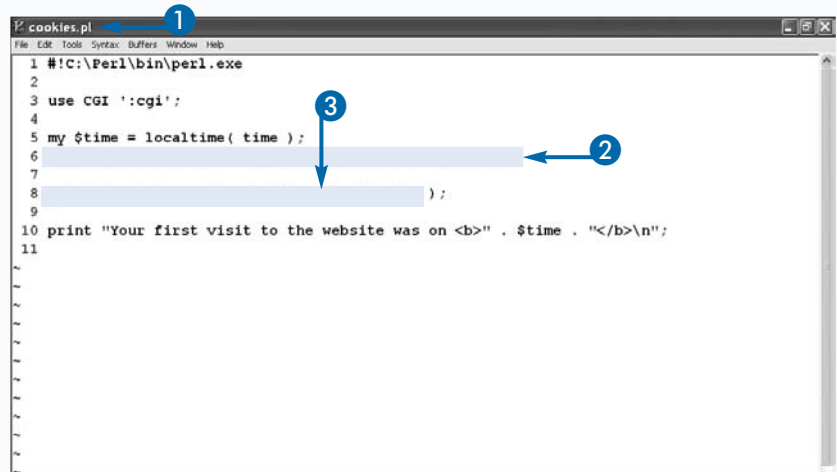
```
VAR = cookie( -attr => value, ... );
```

Once all cookie variables have been assigned, the header routine accepts the cookie objects and relays them to the Web browser. After the header routine runs, no more cookies can be set.

You can set a cookie's value with almost any type of variable, including a scalar, an array, or a hash. This allows you to store a list of values within a single cookie. Applying either the domain or path attributes means that only scripts running under that domain and path can retrieve that cookie's value. You can update a cookie's value by resubmitting another cookie object by the same name. You can delete one by setting its value blank, or to an expiry date in the past.

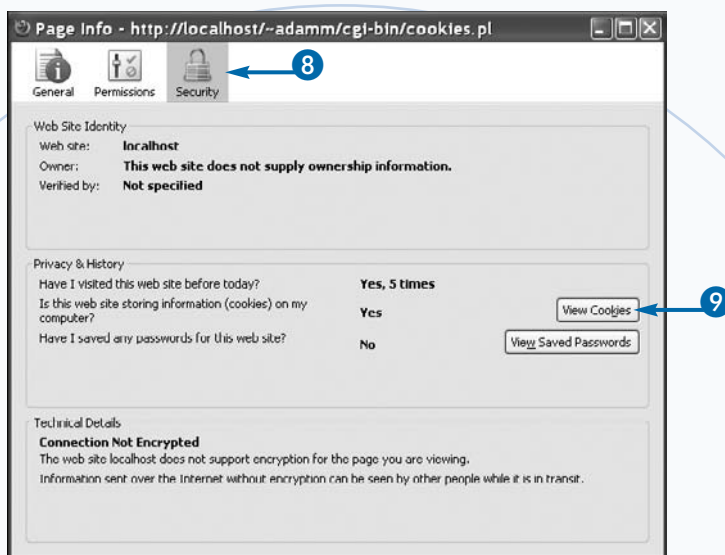
Store HTTP Cookies with the CGI Library

- 1 Open a Perl CGI script in a text editor that uses the CGI library.
- 2 Type `my $var = cookie(name, value);` to store a new cookie into `$var`.
- 3 Type `-cookie => $var` as an argument into `header`.
- 4 Save the CGI script.
- 5 Open the CGI script in a browser.
- 6 Right-click the Web page.
- 7 Click View Page Info.



The Page Info dialog box opens.

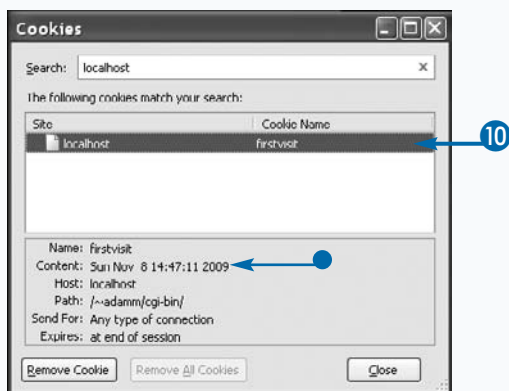
- 8 Click Security.
- 9 Click View Cookies.



The page's cookie summary window opens.

- 10 Select the cookie by its name.
 - The cookie's content.

Note: In this example, you are assigning the cookie a value every time the page is loaded. You should not do this if the cookie already exists. For more information, see the section, "Retrieve HTTP Cookies with the CGI Library."



Apply It

You can assign cookies on every page load, but only before you run the `header` function. If the CGI is reloaded, an established cookie does not need to be resubmitted if its value has not changed. When setting an expiry date, you can use an absolute value for the date, or use a relative value with a plus sign, the date value, and "h", "d", "M", and "y" for hours, days, months, and years.

TYPE THIS

```
my $cookie = cookie( -name => "data",
    -value => \%data,
    -expires => "+3M" );
print header( -cookie => $cookie );
```

RESULTS

The `%data` hash is serialized and stored as the cookie "data" on the user's browser, and will expire in three months. See the section, "Retrieve HTTP Cookies with the CGI Library," for an example of reconstructing the hash and accessing its contents.

Retrieve HTTP Cookies with the CGI Library

The ability to retrieve an HTTP cookie allows your code to remember users as they progress from one Web page to another. You can use the CGI library instead of manually parsing the `$ENV{ 'HTTP_COOKIE' }` environment variable. This is now handled by the `cookie` routine.

To read a cookie, import the CGI library, and then call the `cookie` routine with the cookie's name as an argument. Store the results into a scalar variable. If you want to use the routine as a function, specify the `:cgi` function group when loading the CGI library:

```
VAR = cookie( name );
```

Because the `cookie` routine can handle both reading and writing cookies, if you do not supply a second argument as a value, it automatically assumes that you want to

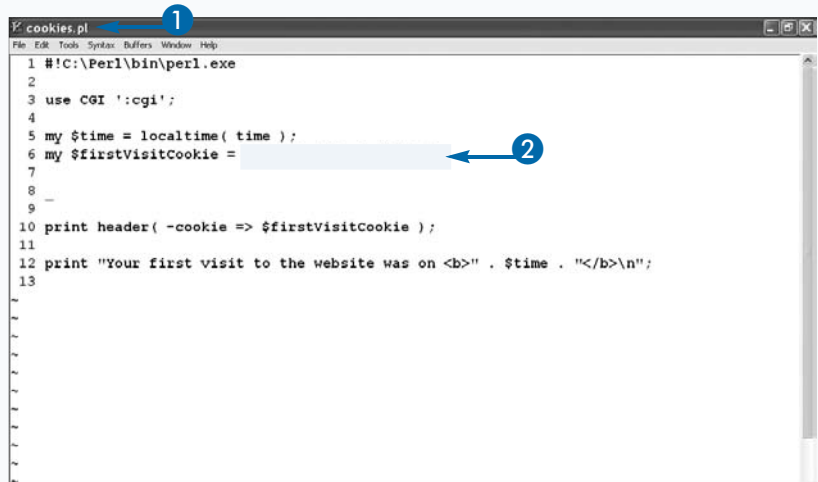
read cookie data. Also, reading a cookie's value can happen at any point in the script. This is unlike writing a cookie's value, which must happen prior to using the `header` function.

Because it is possible to write a cookie using an array or hash variable as its value, you need to be careful when reading it back so that you use the correct receiving variable type. The `param` routine does not warn you if you try to read an array-cookie's value into a scalar. Running `cookie` without any parameters returns a list of all available cookie names.

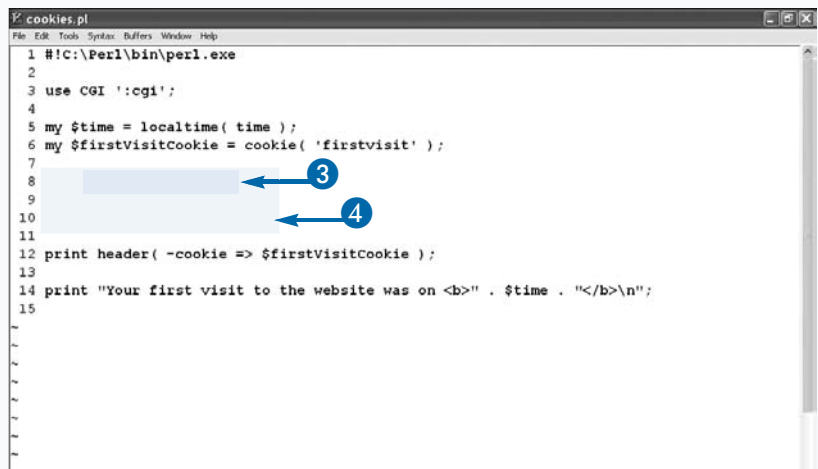
If a cookie has any other attributes, such as an expiry date, domain, or path, that information will not be available when reading the cookie's value. In other words, if the expiry date has elapsed, the domain does not match, or the path is different, then the cookie will simply not exist.

Retrieve HTTP Cookies with the CGI Library

- 1 Open a CGI Perl script in a text editor.
- 2 Type `my $var = cookie(name);` to retrieve a cookie and store it in a variable.
- 3 Start a new conditional test block.
- 4 Use the cookie's variable name as the test expression to verify that the cookie exists.



```
cookies.pl
1 #!C:\Perl\bin\perl.exe
2
3 use CGI ':cgi';
4
5 my $time = localtime( time );
6 my $firstVisitCookie =
7
8
9
10 print header( -cookie => $firstVisitCookie );
11
12 print "Your first visit to the website was on <b>" . $time . "</b>\n";
13
```




```
cookies.pl
1 #!C:\Perl\bin\perl.exe
2
3 use CGI ':cgi';
4
5 my $time = localtime( time );
6 my $firstVisitCookie = cookie( 'firstvisit' );
7
8
9
10
11
12 print header( -cookie => $firstVisitCookie );
13
14 print "Your first visit to the website was on <b>" . $time . "</b>\n";
15
```

- 5 If the cookie does exist, update the main data variable with the cookie's value.
- 6 If the cookie does not exist, use the main data variable's autogenerated value to set it.
- 7 Save the CGI script.

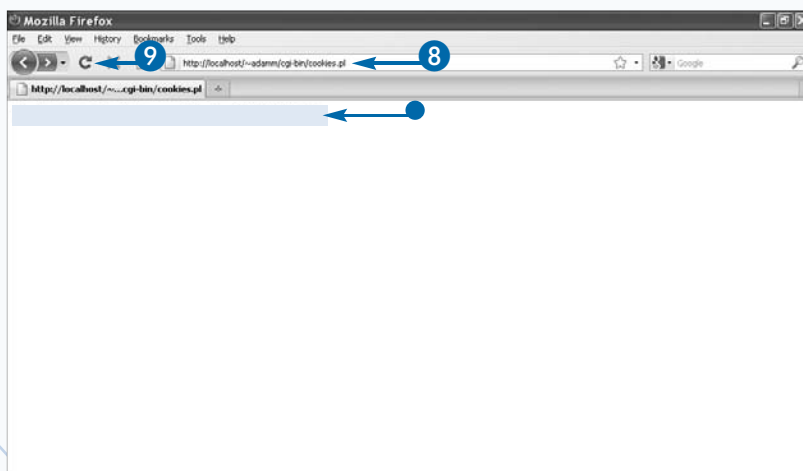
```

1 #!C:\Perl\bin\perl.exe
2
3 use CGI ':cgi';
4
5 my $time = localtime( time );
6 my $firstVisitCookie = cookie( 'firstvisit' );
7
8 if ( $firstVisitCookie ) {
9     # ... (code for existing cookie) ...
10 }
11 else {
12     # ... (code for new cookie) ...
13 }
14
15 print header( -cookie => $firstVisitCookie );
16
17 print "Your first visit to the website was on <b>" . $time . "</b>\n";
18
19
20

```

- 8 Open the CGI script in a browser.
 - The cookie's value displays.
- 9 Click  to reload the Web page.

The displayed value does not change because the cookie's value does not change.



Apply It

The only cookie data accessible by your CGI script is the cookies you initially stored. It is impossible for your CGI script to access cookies from other Web sites, and vice versa. When using the CGI library, the `cookie` routine does all of the reading and writing work for you, but your script still needs to interpret the returned data.

TYPE THIS

```
warn "Cookie names: " .
    join( ' ', cookies() );
```



RESULTS

When you use `cookie` with no arguments, it returns all available cookie names as an array. The array is joined into a comma-separated text string and recorded into the Apache log file through the function `warn`.

TYPE THIS

```
my %data = cookie( "data" );
warn "Cookie data value:
    $data{ KEY }";
```



RESULTS

The cookie "data" has its value loaded into a hash called `%data`. The key/value pairs within the hash are accessible, just like any other variable. The cookie's value is recorded into the Apache log file.

Return Useful Error Messages with CGI::Carp

You can manually place log messages in a Perl CGI script to visually examine the activity on the Apache Web server; however, no timestamp or filename is recorded in the Apache logs. Also, if something in the script goes legitimately wrong, only a vague “500 Internal Server Error” displays in the browser. You can use the CGI::Carp module to solve both of these problems.

CGI::Carp is a stand-alone module that is not considered part of the standard CGI library; instead, it is a subclass of it. This means that its functionality complements the CGI library’s functionality to the point that both are very useful tools for CGI scripting in Perl. You can import the CGI::Carp module into your script just like any other module, except there are no objects to initialize, and no additional functions to use:

Return Useful Error Messages with CGI::Carp

- 1 Open a Perl CGI script in a text editor.
 - 2 Type `use CGI::Carp;`
 - 3 Type `warn "Text";` to write to the Apache log.
 - 4 Type `die "Text";` to write to the Apache log and force-quit the program.
- Note:** Even if you do not use `die`, a Perl syntax error has the same effect with CGI::Carp.
- 5 Save the Perl CGI script.

- 6 Load the CGI script in a browser.
 - A generic error message displays.

Note: If `die` is called after `header`, the user only sees a blank Web browser screen.

```
use CGI::Carp;
```

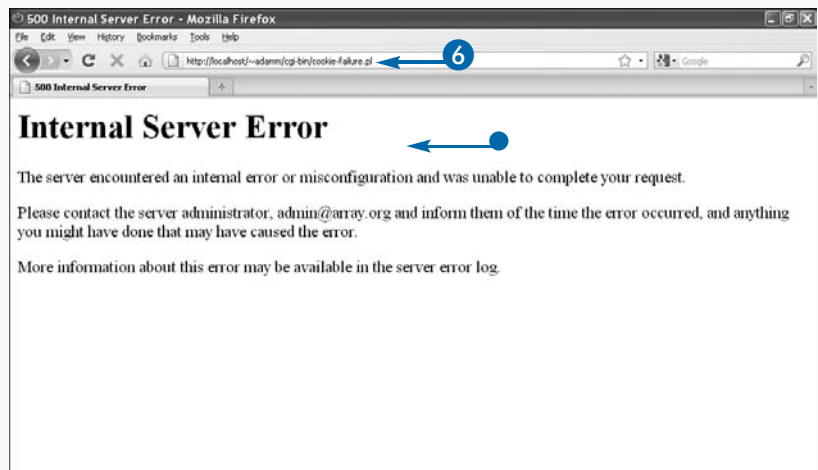
This extends the built-in `warn` and `die` functions so they provide a timestamp, filename, and line number, alongside the actual message.

To enable the HTML-formatted error log messages to be shown to the browser in the event of a failure, use the `fatalToBrowser` argument when importing the module:
`use CGI::Carp qw(fatalToBrowser);`

When you use `die` now, or when Perl has a syntax or internal error, a useful message is sent to the browser citing a “Software error,” followed by the Perl error message, filename, and line number.

CGI::Carp provides other functions, including the amusingly named `carp`, `croak`, `confess`, and `cluck`. For the full documentation, run the PerlDoc command, `perldoc CGI::Carp`.

```
1 #!C:\Perl\bin\perl.exe
2
3 use CGI::Carp;
4
5
6 my $time = localtime( time );
7 my $firstVisitCookie = cookie( 'firstvisit' );
8
9 if ( $firstVisitCookie ) {
10     $time = $firstVisitCookie;
11 }
12
13 else {
14     $firstVisitCookie = cookie( 'firstvisit', $time );
15 }
16
17
18 print header( -cookie => $firstVisitCookie );
19
20 print "Your first visit to the website was on <b>" . $time . "</b>\n";
21
```



7 Open the Apache error log file.

- The warn message is logged.
- The die message is logged.

Note: If you do not use `CGI::Carp`, these functions still produce log entries, but the timestamp, filename, and line number will be gone.

```

[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] HTTP_COOKIE = firstvisit=Sun Nov 8
14:47:11 2009 at C:/Documents and Settings/adamm/My Documents/My
Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] REMOTE_ADDR = 127.0.0.1 at
C:/Documents and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] HTTP_KEEP_ALIVE = 300 at C:/Documents
and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] SERVER_PROTOCOL = HTTP/1.1 at
C:/Documents and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] PATH =
C:/Perl/site/bin/C:/Perl/bin/C:/WINDOWS/system32/C:/WINDOWS/C:/WINDOWS/system32/
Wbem at C:/Documents and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line
9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] REQUEST_URI =
/-adamm/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] GATEWAY_INTERFACE = CGI/1.1 at
C:/Documents and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] SERVER_ADDR = 127.0.0.1 at
C:/Documents and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] DOCUMENT_ROOT = C:/Program
Files/Apache Software Foundation/Apache2.2/htdocs at C:/Documents and Settings/adamm/My
Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Mon Nov 09 03:14:09 2009] [error] [client 127.0.0.1] HTTP_HOST = localhost at C:/Documents
and Settings/adamm/My Documents/My Website/cgi-bin/hello_world.pl line 9.\r
[Thu Mar 25 00:20:59 2010] [error] [client 127.0.0.1] Premature end of script headers:
cookie-failure.pl
  
```

Go back to the Perl CGI script in the text editor.

8 Type `qw(fatalToBrowser)` as a module argument.

9 Save the Perl CGI script.

```

1 #!C:/Perl/bin/perl.exe
2
3 use CGI ':cgi';
4 use CGI::Carp qw(fatalToBrowser);
5
6 my $time = localtime( time );
7 my $firstVisitCookie = cookie( 'firstvisit' );
8
9 if ( $firstVisitCookie ) {
10     $time = $firstVisitCookie;
11     warn "There is a returning visitor to the site!"
12 }
13 else {
14     $firstVisitCookie = cookie( 'firstvisit', $time );
15 }
  
```

10 Refresh the CGI script in the browser.

- The die message displays in the Web browser.

Software error:

For help, please send mail to the webmaster (admin@my.org), giving this error message and the time and date of the error.

Note: The same warning and error messages still show up in the Apache error log.

Extra

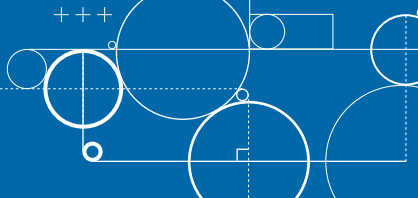
You can customize the `CGI::Carp` module's `fatalToBrowser` HTML error message. For example, you can tailor it to look just like your Web site's layout, and even change the wording of the error message text. This can be useful if you want to provide the user with instructions on how to report the error back to you.

The argument `warningsToBrowser` is similar to `fatalToBrowser`, except when `warn` is called, its text appears as an HTML comment; you can only read it by opening up the Web page's source code in the browser.

Regarding the other available functions, `carp`, `croak`, `confess`, and `cluck`, `carp` performs exactly the same as `warn`, and `croak`, `confess`, and `cluck` are the same as `die`. The difference is that each function interprets who is at fault differently. For more information on how they work, see the `Carp` module's documentation.

The `cluck` function has one additional benefit: it appends a *stacktrace dump* whenever you use it. The Apache error log summary lists all subroutines that were running, right up to `cluck`. To use this feature, you must add the `cluck` option into `CGI::Carp`'s arguments.

Understanding the Benefits of Separating HTML from Perl



By separating HTML from Perl, you can create HTML-only templates solely for displaying syntax, and leave Perl to handle the Web site logic. If you continue to embed HTML in Perl for complex HTML code, such as a table, understanding how that table is constructed could become very difficult. This is especially

true if your table is generated from an external source, if the table has a syntax error, or if someone new is looking at the code. Suddenly, if the table needs to be changed, more effort is spent to understand how the table is built than it takes to apply the change. You can avoid this extra work if you separate the HTML layout from the Perl logic.

Design Goals

The purpose of this design method is to make the overall development effort easier. You can save time creating, debugging, updating, and deploying dynamic Web sites.

Code Simplification

The ordering of the Perl code should not affect the output on the Web browser. This means your code should generate the content using whatever method is most efficient. You should not have to worry about layout ordering, especially when your output requires nested HTML tags such as tables.

Separate HTML and Perl Files

The Perl script should be in charge of collecting the data, and the HTML template file in charge of displaying it. Problems with the HTML layout should not affect the Perl logic that wants to display it, and vice-versa. If the HTML is in a different file, you do not need to change the Perl code to address an HTML-specific problem.

Easier to Create Web Content

When creating a Web page in a graphical HTML composition program, you have to convert its HTML output into Perl code line-by-line. This can be a time-consuming process. Instead, you can simply save the HTML output as a template file, and instruct Perl to use it. You will still need to change the HTML code that will hold dynamically generated content (code that Perl variables will provide) into special HTML-like tags that Perl understands.

Understanding the Big Picture

If someone new works on your Web site, they must read the Perl script and mentally reconstruct the output layout controlled by the printed HTML, display logic, and functionality, all in a single file. When you move the HTML into a separate file, they can read the layout from top to bottom as correct HTML syntax. There will not be large, distracting chunks of Perl code in between the HTML.

Benefits

Once you implement the template infrastructure in Perl, several tasks become easier, such as the development, maintenance, and manageability of the Web site in general.

Convert Mockups into Usable Web Sites

Creating an HTML mockup version of your Web site can help you visualize its layout from a creative perspective. Converting the mockup HTML into a Web site can be easier if you save the mockup as a template. Within the template, you must still identify which portions of the page need to be generated dynamically with Perl, usually done with some sort of template-specific syntax, but the majority of HTML code in your template should be static.

Generate Common HTML from One Template

HTML code that appears on multiple pages should all be sourced from the same template. This can be very useful for toolbars, headers, and footers, or other portions of the Web page that need to be consistent across the entire site. This would mean instructing a single HTML template to reference other templates. If your toolbar ever changes, you simply update the code in the toolbar template once and save it; the entire site reflects the change immediately.

Benefits *(continued)***Better Support for Client-Side Languages**

Just because your separate file is designated as an HTML template does not mean that it only stores HTML. You can write other client-side languages, such as JavaScript, into your HTML template. You can even allow portions of the JavaScript code to support template-generated parameters, using the exact same parameter syntax that your HTML code uses. This more advanced technique allows Perl to control the JavaScript directly, through the HTML template file.

Create Multiple Page-Layout Styles

A single template should define a single layout style: toolbar on the left, text in the middle, logo in the top right, and so on. However, if your users want the toolbar and logo on the right and the text on the left, simply create a new template that rearranges the elements from the original. You need an infrastructure to allow users to select which site template they want to use; the template preference could be stored as a cookie. Both templates must use the same parameter fields, and so the Perl code does not require any change.

Extending to Other Languages

Including raw native-language text in the Perl code may be convenient now, but it will become very difficult to find, extract, and replace later if you want to support multiple languages. Instead, in your HTML template, replace all of your English sentences and standalone words with unique keywords. Create a new lookup file specific to each language that contains the keyword and original language text. From here, the same template infrastructure Perl is using to generate HTML can be used to generate English text, and your HTML template becomes language-independent.

Support Non-HTML Formats

You can use the template model on any file format where dynamic content needs to be generated into a new file. This can include formats such as XML, LaTeX, CSS, and even Perl or C source code.

Implementations

There are multiple implementations available to Perl that follow this HTML template ideology. Choosing which one is right for you is a matter of personal preference.

CGI Library

The CGI library contains built-in functions that generate raw HTML code within Perl scripts. Unfortunately, these CGI functions still need to be used in the same logical order as the original HTML.

Template Library

The Perl Template library is an interface to the Template Toolkit, a Unix-based, command-line-driven program that replaces keywords within a file with generated data. Although the Template library does address all of the design goals mentioned, it follows its own unique syntax, which can be confusing.

HTML::Template

The Perl module HTML::Template follows an HTML-like tagging syntax. You can open the raw template files directly in a browser. The new tags, such as `<tmpl_if>` and `<tmpl_var>`, are simply ignored until your Perl CGI script processes the template and its fields are populated. HTML::Template has been in development since 2000, and is available on CPAN or as a Debian, Red Hat, and ActivePerl PPM package.

Introducing the Perl HTML::Template Module

The HTML::Template module is a third-party library that is designed to separate HTML code from Perl code. You can use it to simplify complex Perl scripts, to re-use common HTML code, and to handle conditional tests and loops in the actual HTML.

Installing HTML::Template

You can install the HTML::Template module directly over CPAN, or using a specific package depending on your Perl distribution system.

PERL SYSTEM	PACKAGE NAME
Debian/Ubuntu	lib-html-template-perl
Red Hat	perl-HTML-Template
ActivePerl PPM	HTML-Template

For instructions on how to use each system format, see Chapter 9. To simply install the module over CPAN, run `cpan HTML::Template` at the command prompt.

HTML::Template does require background knowledge of HTML syntax and usage. Although it is possible to generate an HTML template using a graphical WYSIWYG editor, you must manually edit its code output to add in the specific commands that make dynamic content appear.

Using HTML::Template

To import the HTML::Template module into your Perl script, simply reference the module name at the start of your program:

```
use HTML::Template;
```

After the module has been imported, there are three components where your script interacts with the module: your script must load the template file into an object reference, populate it with parameters, and print the results.

Perl CGI and Template Files

To utilize HTML::Template, you must remove each CGI script and create new template files.

CGI Code Changes

You should remove all of the original HTML code and `print` statements from the CGI script. The only time you will need to use `print` is at the very end of the script, when printing the CGI header, and parsed HTML::Template output. Naturally, this restriction applies only to data going to Apache and the Web browser via standard-output (STDOUT). Printing to other file handles, such as standard-error (STDERR), is still okay.

Template Files

The template files contain the actual HTML code, along with special tags that are specific to HTML::Template. You should

save all template files in their own directory, outside of the CGI directory. The directory may not be Web-accessible, but it should be a unique path for each domain or Web site.

The naming convention of the template file should match the Perl CGI script using it, except for the `.tmpl` extension. If a CGI script has multiple distinctively different HTML screens, use multiple templates. You should suffix its files with an identifier. For templates that you use on multiple pages, choose a unique site-wide prefix, such as `site-header.tmpl` and `site-navbar.tmpl`. Note that you can nest template files to include other templates. This becomes useful when creating shared HTML code across the entire site.

Speed Enhancements

While HTML::Template is a mature and well-tested module, users have noted that its internal code is not very efficient. Other Perl developers have re-implemented HTML::Template's syntax as new modules to obtain

improvements in speed and functionality. These modules are separate projects from the original HTML::Template module, and you must install them manually using CPAN, DEB, RPM, or PPM methods.

Perl Syntax

The HTML::Template follows an object-oriented interface. This means the module is initialized into an object reference variable, which is used to access the module's internal routines as methods.

Load the Template

When the module is initialized with the `new` function, the template file is provided as an argument. You should reference the template file using its full path, or a path relative to the CGI script:

```
my $tmpl = new HTML::Template( filename =>
    TEMPLATE, ... );
```

You may use optional arguments when creating a new template object. This includes additional features such as syntax checking, caching, debugging, and a pre-processing filter.

Populate Template Parameters

The template contains special HTML-like tags that HTML::Template replaces with data supplied by the Perl script. This data is provided with the `param` method:

```
$tmpl->param( KEY => VALUE );
```

This replaces all instances of `<tmpl_var name=KEY>` with `VALUE` in the template file when the browser displays the file.

For the full list of template parameters, see the section, "Understanding the Structure of an HTML::Template File."

Display the HTML

After the template's parameters have all been populated, and all supporting Perl logic has been called, the last thing the CGI script should do is print the CGI header and generated template's output:

```
print header();
print $tmpl->output;
```

There should not be anything else that your script is doing after this point, except for final memory-cleanup tasks.

Extra HTML::Template Features

After you have implemented an HTML::Template Web site, you can enable additional features such as error detection and caching. For the full list of extra features, see the module's documentation.

Error Detection

Enabling the various error-detection options instructs HTML::Template to be either more or less strict on the template syntax as it runs. This can be useful when developing a new template if some parameters are not showing up where you expect. The `die_on_bad_params` option ensures that all parameters used in the Perl CGI go to valid keys in the template file; otherwise, the program quits. Using this feature forces you to generate a template that utilizes all parameters being sent to it. Disabling `die_on_bad_params` is recommended if you routinely assign parameters using shared code that, depending on the template file, may or may not be used. The `strict` option ensures that all tags in the template file that resemble valid HTML::Template tags are correctly formatted.

Caching Templates

Several options exist to enable caching of template files in HTML::Template. Depending on the implementation and environment, this can yield speed improvements of 50 to 90 percent, according to the module's author. The `shared_cache` and `file_cache` options are useful if `mod_perl` is not enabled. The `shared_cache` option uses the `IPC::SharedCache` module, which allows multiple instances of the CGI script to use the same region in memory; this is very useful for high-load Web servers, as it requires less RAM. The `file_cache` option uses the `Storable` module, which serializes the template into a file on the server. Both options reload the template if the file has changed, but are not as efficient as the `cache` option with `mod_perl`.

Documentation and Support

You can access the HTML::Template's manual using the PerlDoc program on the command-line:

```
perldoc HTML::Template
```

This provides you with a detailed explanation of the module's purpose, intention, background, and all the advanced features it supports. An online tutorial is available at <http://html-template.sourceforge.net/article.html>.

Understanding the Structure of an HTML::Template File

The template files that are parsed by HTML::Template must follow a specific tagging format. These tags are replaced with dynamic content sourced by the Perl script to form the actual HTML output. The completed HTML content is then sent to the user's Web browser.

The template elements used by HTML::Template are formatted like HTML elements; they begin with a start tag and finish with an end tag. Some elements use optional

attributes in the start tag, and others do not require an end tag at all. All tag names begin with `tmpl_` followed by the template tag type:

```
<tmpl_TYPE [ATTR=VALUE, ... ]> ... </tmpl_TYPE>
```

HTML::Template supports nesting template elements together. This is useful when looping the same HTML content multiple times for an array (just like the `foreach` block in Perl) and for creating simple conditional tests within the actual HTML (just like an `if` block).

Template Syntax

The template element syntax is case-insensitive, and you can use it anywhere inside a template file. Each element performs a function that is specific to the template and the controlling Perl CGI script.

<TMPL_VAR>

This is the simplest and most common template element. You can use `TMPL_VAR` to supply data provided by the `param` function in the Perl CGI into the final HTML output:

```
$tmpl->param( KEY => VALUE, ... );
```

The `TMPL_VAR` element uses an attribute called `name` that references `KEY`. When the template is output, any instances where `TMPL_VAR`'s name matches `KEY` are replaced by `VALUE`. If `VALUE` is blank, or not defined, the template element is simply omitted from output:

```
<tmpl_var name=KEY>
```

You can use the optional attribute `escape=MODE` to escape `VALUE`'s text for use within HTML form values, JavaScript code values, or URL population. This depends on whether you set `MODE` to `HTML`, `JS`, or `URL`, respectively.

The optional attribute `default=VALUE` allows you to set a default value to the element if `KEY` is undefined.

The `TMPL_VAR` element does not require an end tag.

<TMPL_IF> ... </TMPL_IF>

The `TMPL_IF` element allows you to establish HTML code that should only be used if a Boolean test is true. It also uses the `param` function, but it only displays its content in the HTML output if the data is true:

```
$tmpl->param( KEY => VALUE, ... );
```

The `TMPL_IF` element uses an attribute called `NAME` that references `KEY`. When the template is output, if `KEY` has a defined value, and it is not a literal zero, then `CONTENT` appears:

```
<tmpl_if name=KEY>
  CONTENT
</tmpl_if>
```

The `TMPL_IF` element requires an end tag to signify the end of the `TMPL_IF` block.

<TMPL_ELSE>

`TMPL_ELSE` is an optional element that you can use anywhere within `TMPL_IF`'s content block. It separates and controls HTML code which only appears when `TMPL_IF` is true, and data HTML code which only appears when `TMPL_IF` is false.

This element takes no arguments, and does not use an end tag.

<TMPL_UNLESS> ... </TMPL_UNLESS>

`TMPL_UNLESS` is exactly like `TMPL_IF`, except that it reverses the Boolean test expression. In other words, `KEY` must be undefined, or a literal zero, in order for `CONTENT` to appear.

This takes the same arguments as `TMPL_IF`, and requires a matching end tag. `TMPL_ELSE` is also valid within a `TMPL_UNLESS` block.

<TMPL_LOOP> ... </TMPL_LOOP>

TMPL_LOOP is by far the most complex and powerful tag. It allows you to repeat HTML content multiple times, and provides a new set of TMPL_VAR variables on each pass, just like a foreach or while loop in Perl.

From the perspective of your Perl script, when these commands are passed into param, their value must be represented as a two-dimensional array reference and hash reference. The array reference implies the number of passes the loop makes; the hash reference contains the specific set of data available to each pass's group of TMPL_VARS:

```
$tmpl->param( KEY => [
    { SUBKEY => VALUE1, ... },
    { SUBKEY => VALUE2, ... },
    ...,
] );
```

Note the opening square bracket after KEY. This is the beginning of an anonymous array reference. Each entry in the array indicates a set of subkeys and values that are only valid to that particular entry in the series. When the template is output, the CONTENT within the TMPL_LOOP element displays once for every hash reference in the array.

The data for SUBKEY is first VALUE1, and then VALUE2, depending on which pass of the loop is being shown:

```
<tmpl_loop name=KEY>
  CONTENT <tmpl_var name=SUBKEY>
</tmpl_loop>
```

If KEY is not defined, then CONTENT is never shown, just like TMPL_IF.

This element requires an end tag to signify the end of the TMPL_LOOP block.

<TMPL_INCLUDE>

The TMPL_INCLUDE element allows you to import external template files into the current file. All elements within the new file are acted upon with the same parameters provided when the original was called from Perl. The only argument it takes is the external filename:

```
<tmpl_include name=FILENAME>
```

If a newly included template includes more templates, then they are also imported. This element does not use an end tag.

Caveats

When creating templates, there are some common pitfalls that are very easy to avoid if you are aware of them.

Missing End Tags

If you happen to forget a </TMPL_IF> somewhere in the middle of a template, your CGI script reports a syntax error when you run it. A good way to minimize this risk is to get into the habit of indenting code whose content requires an end tag. This includes existing HTML tags:

```
<table>
  <tmpl_loop name=rowList>
    <tr>
      <td><tmpl_var name=rowNumber></td>
      <tmpl_if name=rowValue>
        <td><tmpl_var name=rowValue></td>
      </tmpl_if>
    </tr>
  </tmpl_loop>
</table>
```

From this simple example you can see content following <tmpl_loop>, and <tmpl_if> is indented on the next line. Once the content is done, you remove the indent when you use the end tag. This looks exactly the same as opening and closing curly brackets in Perl.

Missing Parameters

If your Perl script specifies a KEY parameter that your template is not using, HTML::Template induces a failure. This situation typically arises when the Perl script and template are at different levels of development. You can silently ignore this by disabling the option die_on_bad_params when creating a new HTML::Template object reference.

Strict Tag Names

If your template misspells an element's tag, such as <tmpl_varrr name=KEY>, HTML::Template induces a failure. You can ignore these checks by disabling the option strict when creating a new HTML::Template object reference.

All element tags that begin with tmpl_ are checked for validity when you enable strict.

Create a New Template File

When creating a new template file for the HTML::Template module, you must lay out your HTML intelligently so that it can take advantage of the module's search-and-replace functionality — replacing template-specific tags with dynamically generated content by Perl — when the Web page displays in a browser. The template creation process involves identifying which parts of the page should be replaced by dynamically generated content, and which parts should remain static. Often, the dynamic content is being sourced from something external, such as a database backend accessed through Perl. The static content is usually the original HTML code that controls the page's layout, fonts, colors, and common text. The dynamic content needs to be replaced by new template elements. Some template elements can also

control blocks of static HTML. For example, you may want to loop some HTML when generating tables or lists, and omit entire blocks of HTML if a conditional test in Perl returns false. The looping aspect allows your template to dynamically grow and shrink, depending on how much data needs to be displayed on the page.

The best way to begin is to manually create a static HTML file, but populate it with mockup data. Use this time to test your mockup in different browsers and operating systems, and to develop any additional client-side code, such as JavaScript. When content with how the mockup looks and works, convert it into a template file and create the Perl CGI script for it. The conversion process happens in two parts: first you replace mockup data with template elements specific to HTML::Template, and then you identify any portions of the HTML code that need to be logically controlled or repeated by Perl.

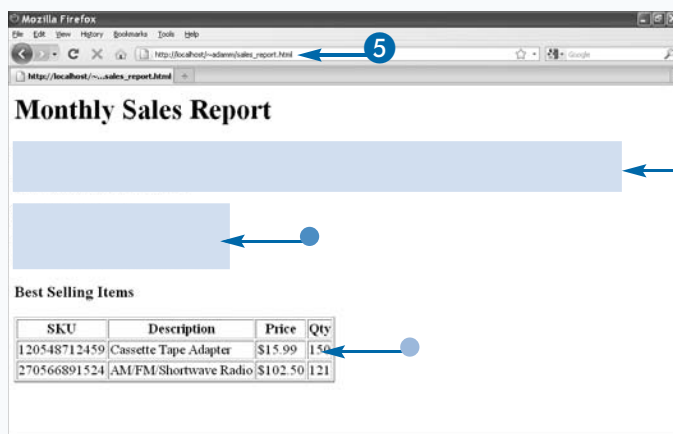
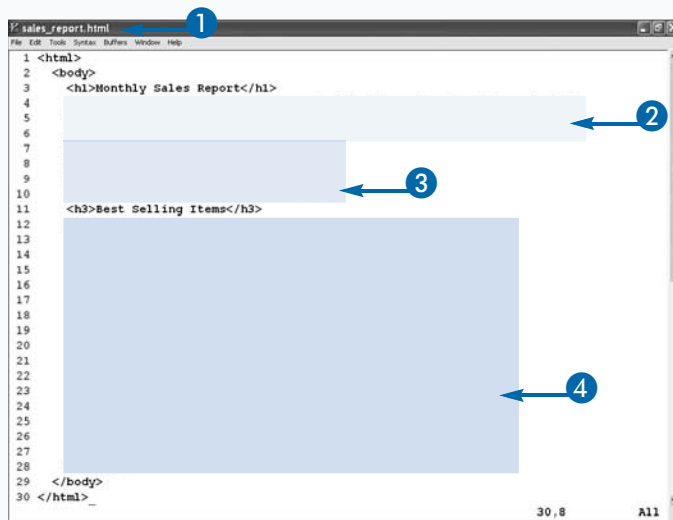
Create a New Template File

- 1 Create a static HTML Web page in a text editor.
- 2 Add a sample text message.
- 3 Add sample data.
- 4 Add a sample table with data.

Note: The sample data text is not important, but its intention and location in the HTML mockup are important.

- 5 Open the HTML file in a browser.

- The sample message.
- The sample data.
- The sample table with data.



- 6 Type `<tmpl_if name=key>` prior to the sample message.
- 7 Type `</tmpl_if>` at the end of the sample message.

Note: The sample message may or may not appear, but the text within stays the same. Perl is able to control this.

- 8 Type `<tmpl_var name=key>` to replace the sample data.
- 9 Type `<tmpl_loop name=loopkey>` at the start of the repeating portion of the table.
- 10 Truncate the table to only include one row in the loop.
- 11 Replace the table cells with more `TMPL_VAR` elements.
- 12 Type `</tmpl_loop>` at the end of the repeating row.
- 13 Save the file with a `.tmpl` extension.

```

1 <html>
2 <body>
3 <h1>Monthly Sales Report</h1>
4 The following sales have been reported in the region for the period of
5 <tmpl_var name=startDate> to <tmpl_var name=endDate>. <p>
6
7 <tmpl_if name=metSalesPlan>
8 <h3>Best Selling Items</h3>
9 <table border=1>
10 <tr>
11 <th>SKU</th><th>Description</th><th>Price</th><th>Qty</th>
12 </tr>
13 <tr>
14 <td>120548712459</td>
15 <td>Cassette Tape Adapter</td>
16 <td>$15.99</td>
17 <td>150</td>
18 </tr>
19 <tr>
20 <td>270566891524</td>
21 <td>AM/FM/Shortwave Radio</td>
22 <td>$102.50</td>
23 </tr>
24 </table>
25 </body>
26 </html>

```

```

5 <tmpl_var name=startDate> to <tmpl_var name=endDate>. <p>
6
7 <tmpl_if name=metSalesPlan>
8 <h3>Best Selling Items</h3>
9 <table border=1>
10 <tr>
11 <th>SKU</th><th>Description</th><th>Price</th><th>Qty</th>
12 </tr>
13 <tr>
14 <td>120548712459</td>
15 <td>Cassette Tape Adapter</td>
16 <td>$15.99</td>
17 <td>150</td>
18 </tr>
19 <tr>
20 <td>270566891524</td>
21 <td>AM/FM/Shortwave Radio</td>
22 <td>$102.50</td>
23 </tr>
24 </table>
25 </body>
26 </html>

```

Extra

You have converted your mockup data into template elements: a message that may appear only sometimes, the data that will be populated, and a table that could have multiple rows of information. A corresponding Perl CGI script will control all of this later.

You should save the actual template filename with a `.tmpl` extension, and, if possible, the filename should also match the Perl script name. For example, the index page of the Web site is generated by the Perl script `index.pl`, which in turn uses `index.tmpl` to generate its output. While this naming convention is not mandatory, it is recommended, especially when your site grows to dozens of pages, each with unique Perl CGI scripts and template files.

Eventually, if you want to support multiple languages on your Web site, you should even convert the raw text in your template into a template variable. This would mean replacing every individual paragraph, sentence, and word as `<tmpl_var>` with a unique key, and then moving the original text and all available translations into language-specific template files.

Import the HTML::Template Module

The HTML::Template module must be imported into any Perl scripts that will be generating HTML content from a template. You can then use the HTML::Template module to process the template file and produce a dynamic Web page.

The HTML::Template module provides its methods in an object-oriented fashion. You must manually install it on each server and workstation that will run Perl scripts that use the module. You may install the module using CPAN, or by a distribution-specific package, after which it must be imported into your Perl CGI scripts and initialized into an object reference variable, usually called `$tmpl`.

It is a good practice to strip all HTML code from the Perl script at this stage and place it all into a separate template

file. Your Perl script should not be printing *anything* as output, except for CGI's header and HTML::Template's output methods. Printing these two methods should be the very last thing your Perl script does before it quits.

It is a good idea to deactivate the `die_on_bad_params` option when initializing HTML::Template. Leaving this option on, which is the default behavior, causes your CGI script to fail if you specify a template parameter whose key is not populated by Perl. This can be frustrating if you are not yet finished writing the Perl script or template, but you want to execute the CGI script and template to ensure the correct development path. Once everything is complete, and your Perl CGI script is using its template correctly, you may re-enable `die_on_bad_params` if you want to be absolutely sure that all template parameters going in have a matching template variable going out.

Import the HTML::Template Module

- 1 Open a Perl CGI script in a text editor.
- 2 Cut all HTML print statements, and paste them into a TEMPL file.

Note: For more information about changing all scalars into `TMPL_VARS`, see the section, "Create a New Template File."

- 3 Delete the `print $cgi->header()` statement. It will be restored after the template is loaded.
- 4 Type `use HTML::Template;`
- 5 Type `my $tmpl = HTML::Template->new()` to create a new template object.

- 6 Type `filename => "template.tmpl"`.

Note: If the template file does not exist in the same directory as the CGI script, you need to provide an absolute or relative path to it.

- 7 Type `die_on_bad_params => 0,`
- 8 Type `;` to close the `new` constructor function.

```
1 #!c:/perl/bin/perl.exe
2
3 use CGI;
4
5 my $cgi = new CGI;
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
1 #!c:/perl/bin/perl.exe
2
3 use CGI;
4
5
6
7 my $cgi = new CGI;
8
9
10
11
12
13
```

- 9 Restore the CGI header print statement at the bottom of the file.
- 10 Type `print $tmpl->output();` on the last line.
- 11 Save the CGI script.

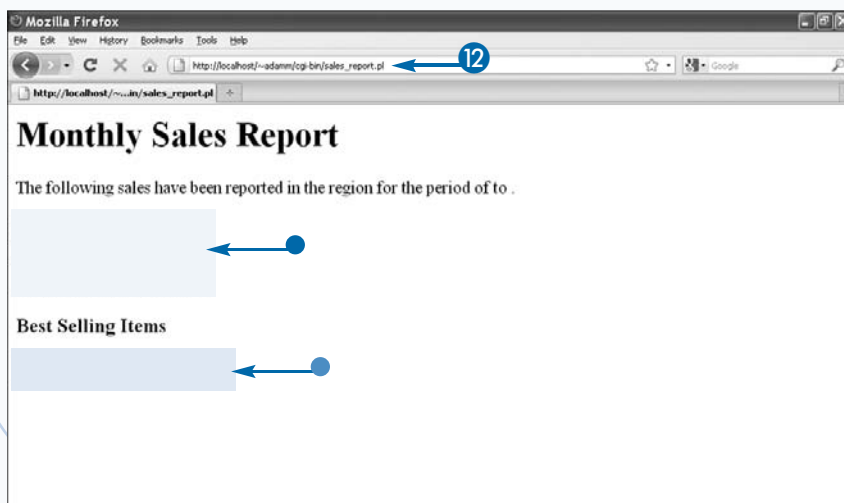
```

1 #!C:/perl/bin/perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6
7 my $cgi = new CGI;
8 my $tmpl = HTML::Template->new(
9     filename => "sales_report.tmpl",
10    die_on_bad_params => 0,
11 );
12
13
14
15
16
17
18

```

- 12 Open the CGI script in a browser.

- The text fields are blank, awaiting real data.
- The table is blank.



Extra

When initializing the `HTML::Template` object reference, you demonstrated using the options `filename` and `die_on_bad_params`. You can supply other optional attributes here that further affect how `HTML::Template` works. Here are some of the more popular options available; you can find the full list in the module's documentation.

OPTION ATTRIBUTE	PURPOSE
<code>associate => \$cgi</code>	Automatically import <code>\$cgi</code> 's HTML form input as template parameters.
<code>debug => 1</code>	Write <code>HTML::Template</code> debug information to the Apache error log.
<code>global_vars => 1</code>	Allow vars declared outside of loops to be available inside.
<code>path => [DIRS]</code>	Specify directory paths to search for the template file.

You can actually call `HTML::Template`'s `param` method at any point in the code after `$tmpl` is declared, but before output is used. This means that you can use the Perl logic to start generating data to display it to the user at any time between these two points. In this example, the table data parameter is hard-coded through a nested array and hash reference. Realistically, you are not going to generate tables this way.

Display Data with TMPL_VAR

You can display actual data in the template using the `TMPL_VAR` element. To define text that should be displayed in the template, a unique key needs to be assigned to each piece of data that `TMPL_VAR` will represent. In the Perl CGI script, you use the `param` method to define each key's value:

```
$tmpl->param( KEY => VALUE );
```

To display the text in the template, the `TMPL_VAR` tag element must contain at least one attribute called `name`. In the `TMPL` file, the `name` attribute's `KEY` is the same key, which was used with `param`:

```
<tmpl_var name=KEY>
```

This is all that is required when generating *normal text* intended for the final Web page output. Normal text is content the user cannot modify when it appears in their

Web browser. Text the user can modify usually appears inside of HTML form input fields. Displaying this data as a field's default value requires a special `escape` attribute:

```
<input type=text name=KEY value="<tmpl_var name=KEY escape=html">">
```

Using `escape=html` slightly alters the data being sent by Perl. The characters `"`, `<`, `>`, and `&` are replaced with a format that is safe for form input values. This ensures that any data entered by the user cannot affect the HTML around it. Similarly, you can use `escape=js` when pre-populating JavaScript variables with data from Perl. You can use the attribute `default` to set the default text for an element. This is useful if the template's `KEY` is never defined in Perl. `TEXT` is displayed, rather than nothing:

```
<tmpl_var name=KEY default=TEXT>
```

Display Data with TMPL_VAR

- 1 Open a template file in a text editor.
- 2 Type `<tmpl_var name=key1>`.
- 3 Type `<tmpl_var name=key1 escape=html>`.

Note: You can reuse the same key in multiple `TMPL_VAR` tags.

Note: You do not need to worry about nesting the double-quotes here; when the template is processed, the browser only sees the input field's quotes, not HTML::Template's quotes.

- 4 Type `<tmpl_var name=key2>`.
- 5 Type `<tmpl_var name=key2 escape=html>` within an HTML input field.
- 6 Save the template file with the `.tmpl` extension.

```
1 <html>
2 <body>
3 <form>
4 <table cellspacing=5 cellpadding=5 border=0>
5 <tr bgcolor=#eeeeee>
6 <th>Field</th><th>Original Value</th><th>Escaped HTML Value</th>
7 </tr>
8 <tr>
9 <td>key1</td>
10 <td><tmpl_var name=key1></td>
11 <td><input type="text" value="<tmpl_var name=key1 escape=html">"></td>
12 </tr>
13 <tr>
14 <td>key2</td>
15 <td><input type="text" value=""></td>
16 <td><input type="text" value=""></td>
17 </tr>
18 </table>
19 </form>
20 </body>
21 </html>
```

```
1 <html>
2 <body>
3 <form>
4 <table cellspacing=5 cellpadding=5 border=0>
5 <tr bgcolor=#eeeeee>
6 <th>Field</th><th>Original Value</th><th>Escaped HTML Value</th>
7 </tr>
8 <tr>
9 <td>key1</td>
10 <td><tmpl_var name=key1></td>
11 <td><input type="text" value="<tmpl_var name=key1 escape=html">"></td>
12 </tr>
13 <tr>
14 <td>key2</td>
15 <td><input type="text" value=""></td>
16 <td><input type="text" value=""></td>
17 </tr>
18 </table>
19 </form>
20 </body>
21 </html>
```


- 7 Open a Perl script that uses the HTML::Template module in a text editor.
- 8 Use the correct .tmpl file as the template.
- 9 Type `$tmpl->param(key1 => "HTML");` to populate *key1*'s value.
- 10 Type `$tmpl->param(key2 => "HTML");` to populate *key2*'s value.
- 11 Save the Perl CGI script.
- 12 Open the Perl CGI script in a browser.
- 13 Press Ctrl+U to view the page's source code.
 - Key1's value appears as HTML.
 - Key1's value appears as literal text.
 - Key2's value appears incorrectly within the form input.
 - Key2's value appears correctly.

```

1 #!C:\Perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new(
8     filename => "
9     die_on_bad_params => 0,
10 );
11
12
13
14
15 print $cgi->header();
16 print $tmpl->output();
17

```

Field	Original Value	Escaped HTML Value
key1	HTML	<HTML>
key2	HTML	"HTML"


```

<table cellspacing=5 cellpadding=5 border=0>
<tr bgcolor=#ffffff>
<th>Field</th><th>Original Value</th><th>Escaped HTML Value</th>
</tr>
<tr>
<td>key1</td>
<td>HTML</td>
<td><HTML></td>
</tr>
<tr>
<td>key2</td>
<td>HTML</td>
<td>"HTML"</td>
</tr>
</table>
</form>
</body>
</html>

```

Apply It

You can relay the `%ENV` hash to supply the template with environment information relevant to the user's CGI session. Text from the environment can now be displayed to the user through the template. This will be especially useful later with HTML forms that are displayed and processed with the same Perl CGI script and template.

TYPE THIS INTO PERL

```

while ( my ( $key, $val ) = each %ENV ) {
    $tmpl->param( $key => $val );
};

```

TYPE THIS INTO THE TEMPLATE

Your IP address is `<tmpl_var name=REMOTE_ADDR>`.

RESULTS

All environment variables are available to the template. The IP address of the user appears in the template output.

Remember to set `die_on_bad_params => 0`. Otherwise, your CGI script will fail unless your template actually uses every environment variable in its output!

Control Template Content with TMPL_IF, TMPL_ELSE

It is possible to control HTML content within the template by introducing a conditional test around one or more HTML tags or text. You can use this to control what displays to the user only if a test returns true, and what displays if it returns false. Before you can define a conditional test in the HTML template, you need something to test it against. You can use all data that is usable as a `<tmpl_var>` variable as a conditional test, performing a basic Boolean test only. In other words, *true* if the variable is defined with content, *false* if defined but with no content or not defined at all:

```
<tmpl_if name=KEY>CONTENT</tmpl_if>
```

The content in between the `TMPL_IF` tags can be anything, including HTML, text, `TMPL_VARS`, or even more `TMPL_IF` blocks. Optionally, you can use a single

`TMPL_ELSE` tag to bisect a `TMPL_IF` block. This allows you to set up content specific to `KEY` returning true, and content for `KEY` returning false:

```
<tmpl_if name=KEY>TRUE CONTENT<tmpl_else>FALSE CONTENT</tmpl_if>
```

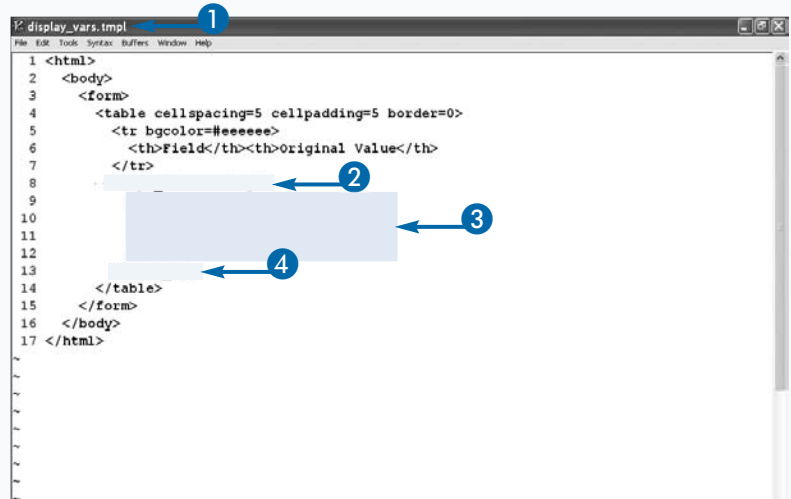
Remember that you can still use the variable tested using `TMPL_IF` within the actual content. This can be extremely useful when a variable is optional, and you need additional static text around the variable to provide content. This is true in the case of error messages:

```
<tmpl_if name=errorMsg>We're sorry, the following error occurred: <b><tmpl_var name=errorMsg></b>. Please resubmit your request and try again.</tmpl_if>
```

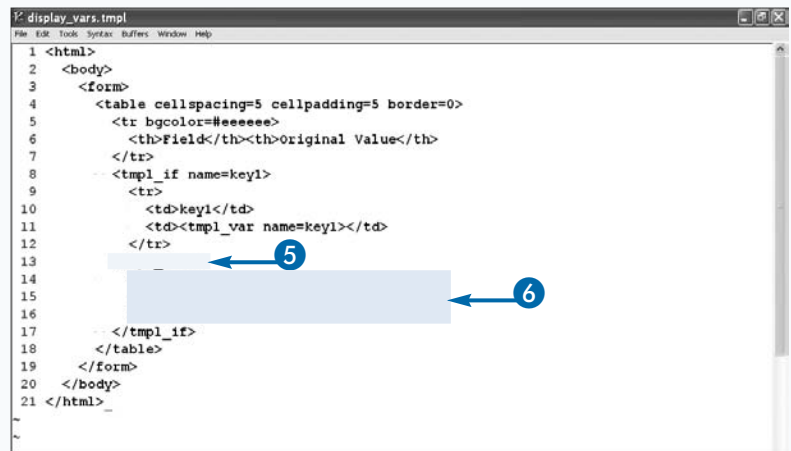
Naturally, there is no point in displaying the *We're sorry* text if no actual error occurred and `errorMsg` is undefined.

Control Template Content with TMPL_IF, TMPL_ELSE

- 1 Open a template file in a text editor.
- 2 Type `<tmpl_if name=key>`.
- 3 Define the content that will only display if `key` is defined.
- 4 Type `</tmpl_if>`.
- 5 Type `<tmpl_else>`.
- 6 Define the content that should only appear if `key` is not defined.
- 7 Save the template file.



```
1 <html>
2 <body>
3 <form>
4 <table cellspacing=5 cellpadding=5 border=0>
5 <tr bgcolor=#eeeeee>
6 <th>Field</th><th>Original Value</th>
7 </tr>
8
9
10
11
12
13
14 </table>
15 </form>
16 </body>
17 </html>
```



```
1 <html>
2 <body>
3 <form>
4 <table cellspacing=5 cellpadding=5 border=0>
5 <tr bgcolor=#eeeeee>
6 <th>Field</th><th>Original Value</th>
7 </tr>
8 <tmpl_if name=key1>
9 <tr>
10 <td>key1</td>
11 <td><tmpl_var name=key1></td>
12 </tr>
13
14
15
16
17 </tmpl_if>
18 </table>
19 </form>
20 </body>
21 </html>
```

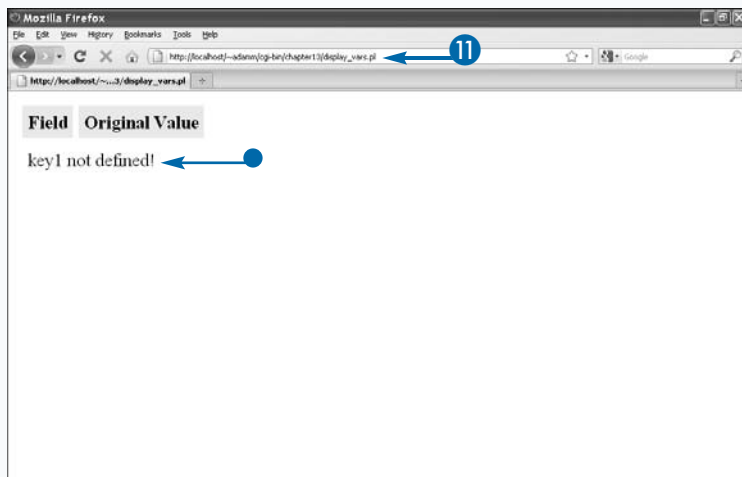
- 8 Open a Perl CGI script in a text editor that uses the HTML::Template module.
- 9 Ensure `key` is not defined as a template parameter.
- 10 Save the Perl CGI script.

```

1 #!C:\Perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new(
8     filename => "display_vars.tmpl",
9     die_on_bad_params => 0,
10 );
11
12
13
14 print $cgi->header();
15 print $tmpl->output();
16

```

- 11 Open the Perl CGI script in a browser.
 - The browser displays the alternate text.



Apply It

The `TMPL_IF` conditional test focuses on passing if its test parameter is true. If you want to test for false parameters, there is no such syntax as `<tmpl_var not name=VAR>`; instead, use `TMPL_UNLESS`.

TYPE THIS INTO THE TEMPLATE

```

<tmpl_unless name=key1>
  key1 not defined!
</tmpl_unless>

```



RESULTS

The text appears if you do not define the `key1` parameter.

There is no `TMPL_ELSEIF` tag. This is the only real feature lacking in the `HTML::Template`, but you can use the same logic by creating a new `TMPL_IF` block immediately following a `TMPL_ELSE` tag. Unfortunately, this can get really messy and confusing if you require more than two instances of *else if* logic in a row.

Repeat Template Content with TMPL_LOOP

You can control and repeat template content based upon a list of variables provided by Perl with the `TMPL_LOOP` tag. You can use this to minimize the amount of unique HTML that you write in the template by working under the condition that each item in the loop appears with the exact same HTML code. This tag is most often used for tables where each table row appears to the user in a similar style:

```
<table>
  <tmpl_loop name=LIST>
    <tr><td><tmpl_var name=KEY></td></tr>
  </tmpl_loop>
</table>
```

Perl uses an array reference to populate `TMPL_LOOP` where each array entry represents one row of data. Within each

entry is another hash reference that is used to populate the keys within that row. This makes each `TMPL_VAR` nested within `TMPL_LOOP` actually have a value.

You can use the `TMPL_IF` tag outside of the `TMPL_LOOP` tags to validate whether the loop has been defined, and actually has content in its list. This is extremely useful to control the leading and trailing HTML code that is related to the loop display code, as this code is not within the actual loop. If your table includes a single top header row, you place that HTML just before the opening `<tmpl_loop>` tag. Naturally, if the loop's array is empty, the entire table, including the table column header row, is omitted.

Be careful about displaying too much data inside of `TMPL_LOOP`. Limit the size of the array forwarded to the template, introducing some sort of paging or searching functionality.

Repeat Template Content with TMPL_LOOP

- 1 Open a template file in a text editor.
- 2 Type `<tmpl_loop name=list>` to begin the template loop.
- 3 Type `<tmpl_var name=key>` to access a parameter within the loop.
- 4 Type `</tmpl_loop>` to end the loop.

```
1 <html>
2 <body>
3   <h1>Display Contacts</h1>
4   <table cellspacing=5 cellpadding=5 border=0>
5     <tr bgcolor=#eeeeee>
6       <th>Name</th><th>Address</th><th>Phone Number</th>
7     </tr>
8     <tmpl_loop name=list>
9       <tr>
10        <td><tmpl_var name=key></td>
11        <td><tmpl_var name=key></td>
12        <td><tmpl_var name=key></td>
13      </tr>
14    </tmpl_loop>
15  </table>
16 </body>
17 </html>
```

- 5 Type `<tmpl_if name=list>` to check if the table exists.
- 6 Type `<tmpl_else>` to specify alternate text.

Note: You use `TMPL_ELSE` if the loop is either not defined, or is defined but its array has no elements.

- 7 Type content to be displayed if the list is empty.
- 8 Type `</tmpl_if>` to close the conditional block.

```
1 <html>
2 <body>
3   <h1>Display Contacts</h1>
4   <table cellspacing=5 cellpadding=5 border=0>
5     <tr bgcolor=#eeeeee>
6       <th>Name</th><th>Address</th><th>Phone Number</th>
7     </tr>
8     <tmpl_if name=list>
9       <tmpl_loop name=contactList>
10        <tr>
11          <td><tmpl_var name=name></td>
12          <td><tmpl_var name=address></td>
13          <td><tmpl_var name=phone></td>
14        </tr>
15      </tmpl_loop>
16    </table>
17    <tmpl_else>
18      <div>No contacts found.</div>
19    </tmpl_if>
20 </body>
21 </html>
```

- 9 Open a Perl CGI script in a text editor that uses the HTML::Template module.
- 10 Type `$tmpl->param(list => [` to begin a new list as an array reference.
- 11 Type `]`; to end the array reference and the `param` statement.
- 12 Type `{` and `}`, to define a hash ref as a single loop of the array ref.
- 13 Type `key => "value"`, for each parameter in the loop.
- 14 Save the Perl CGI script.

```

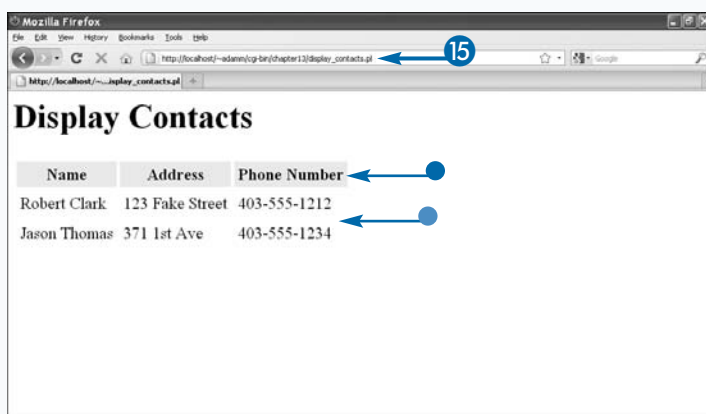
1 #!C:\perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new(
8     filename => "display_contacts.tmpl",
9     die_on_bad_params => 0,
10 );
11
12 {
13     {
14         name => "Jason Thomas",
15         address => "371 1st Ave",
16         phone => "403-555-1234",
17     },
18 }
19
20 print $cgi->header();
21 print $tmpl->output();
22
23
24
25
26
27
28

```

- 15 Open the Perl CGI script in a browser.

- The non-looping HTML text appears.
- The looping HTML content is displayed.

Each loop displays unique content from the array ref in Perl.



Apply It

When creating table rows dynamically, it is possible to declare a new array reference, and then use the `push` function to inject anonymous hash references into it as many times as necessary. Once you do this, send the array reference into the `param` function for the template to process and display.

TYPE THIS INTO PERL

```

my $contacts = [];
push( @{$contacts}, {
    name    => "Robert Clark",
    address => "123 Fake Street",
    phone   => "403-555-1212",
} );
$tmpl->param( contactList => $contacts );

```



RESULTS

One row of the table produced by the template is populated. Repeat the `push` function, with unique content, for as many table rows as necessary.

See Chapter 21 for an example of importing dynamic content from an external source into `TMPL_LOOP` using the `push` function.

Nest Templates with TMPL_INCLUDE

You can simplify maintenance of your Web site by splitting up common template HTML into multiple template files. You can then reference multiple template files when using `TMPL_INCLUDE` on a single Web page.

This technique is most commonly used for Web site headers, toolbars, footers, or any content that is designed to appear on more than one page. In the future, if one of these shared templates needs a change, such as updating a Web site logo or copyright information, you will only need to update a single template file. This will be reflected on every Web page that uses that shared template.

To nest templates with `TMPL_INCLUDE`, you need to use a simple element tag with a `name` attribute:

```
<tmpl_include name=FILENAME>
```

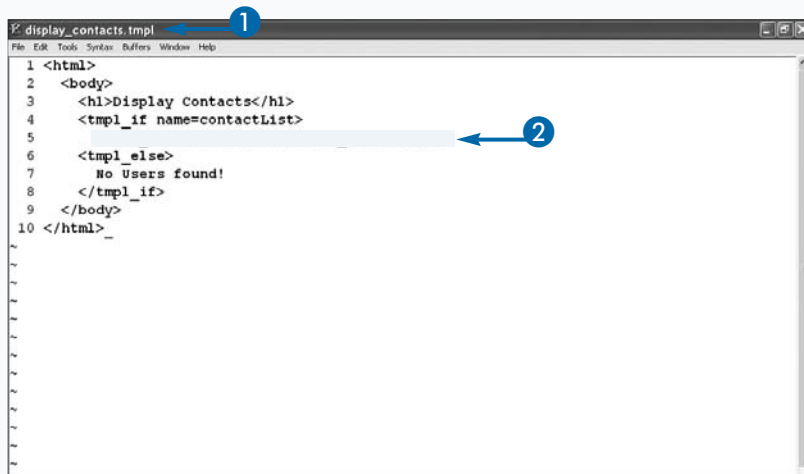
The referenced file can be an absolute path; otherwise, it should be in the same directory as the parent template, or be represented as a relative path. If `HTML::Template` still cannot find the nested template, it consults the server's environment variable `HTML_TEMPLATE_ROOT`, if it is defined. Finally, it checks in the `path` option, which you can set in the Perl CGI when initializing the `HTML::Template` object reference.

```
my $tmpl = HTML::Template->new( ... ,  
    path => [ DIR, ... ] );
```

All parameters defined by the Perl CGI are applied recursively to all included child templates. This allows you to control the content of the included templates using the same template element tags.

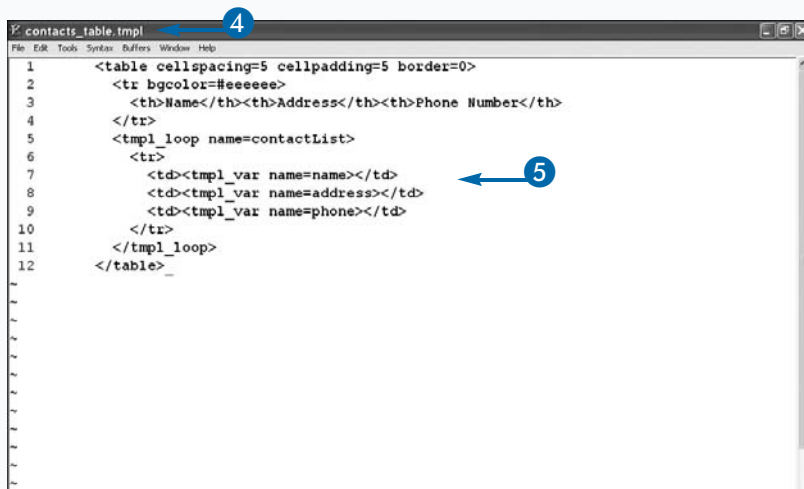
Nest Templates with TMPL_INCLUDE

- 1 Open a template file in a text editor.
- 2 Type `<tmpl_include name="sharedfile.tmpl">`.
- 3 Save the parent template file.



```
1 <html>  
2 <body>  
3 <h1>Display Contacts</h1>  
4 <tmpl_include name="contactList">  
5  
6 <tmpl_else>  
7 No Users found!  
8 </tmpl_if>  
9 </body>  
10 </html>
```

- 4 Create a new template file in a text editor.
- 5 Add some HTML or TMPL content.
- 6 Save the file as `sharedfile.tmpl`.



```
1 <table cellpadding=5 cellspacing=5 border=0>  
2 <tr bgcolor=#eeeeee>  
3 <th>Name</th><th>Address</th><th>Phone Number</th>  
4 </tr>  
5 <tmpl_loop name="contactList">  
6 <tr>  
7 <td><tmpl_var name="name"></td>  
8 <td><tmpl_var name="address"></td>  
9 <td><tmpl_var name="phone"></td>  
10 </tr>  
11 </tmpl_loop>  
12 </table>
```


- 7 Open a Perl CGI script in a text editor that uses the HTML::Template module.
- 8 Use the parent template file as the input template name.
- 9 Save the Perl CGI script.

```

1 #!C:\Perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new(
8     filename => "
9     die_on_bad_params => 0,
10 );
11
12
13 $tmpl->param( contactList => [
14     {
15         name => "Robert Clark",
16         address => "123 Fake Street",
17         phone => "403-555-1212",
18     },
19     {
20         name => "Jason Thomas",
21         address => "371 1st Ave",
22         phone => "403-555-1234",
23     },
24 ] );
25
26 print $cgi->header();
27 print $tmpl->output();
28

```

- 10 Open the Perl CGI script in a Web browser.
 - The browser displays the template parentfile.tmpl.
 - The browser displays the template sharedfile.tmpl where you originally used TEMPL_INCLUDE.



Extra

The `TMPL_INCLUDE` tag is very useful. It is a great way to offload shared content into a single file, and then share that file across multiple templates. However, you do need to be careful if an included template makes another call to `TMPL_INCLUDE`.

HTML::Template has a built-in limit to the number of nested templates it will honor. This prevents recursive loops in case the child template re-includes the parent. The default maximum depth is ten levels, but you can adjust this if necessary with the `max_includes` option.

HTML::Template also makes an effort to cache all template files that are referenced by the Perl CGI script and `TMPL_INCLUDE`. It provides an option to enable one of several algorithms that range in complexity, given the memory available on the Web server.

The simplest method of caching provided will keep each template file active in memory by comparing the original file's modification timestamp on each request. More complex instances involve sharing the cache across multiple Apache threads, or a combination of each technique.

To make caching beneficial, you need to enable `mod_perl` in Apache. See Chapter 23 for more information.

Create an HTML::Template Header and Footer

You can create a persistent header and footer within HTML::Template files by using the `TMPL_INCLUDE` tag, and reference simple TMPL header and footer files. This allows you to standardize static content across an entire Web site out of a pair of TMPL files.

The header and footer HTML code refers not only to visible content that is presented at the top and bottom of every Web page, but also to any HTML code that must come before and after the dynamic body content.

The header template file must introduce the very top of every Web page. This includes the beginning `<html>`, `<title>`, `<style>`, and `<body>` tags. It is also responsible for *starting* the page's global table, including all printable content that appears at the top of every page.

The footer template file must close off the very bottom of every Web page. Therefore, in this order, it is responsible for *ending* the page's global table, and displaying any of the site's fine print, such as a copyright claim and a privacy policy link. Finally, you complete the page with the ending `</body>` and `</html>` tags.

The global table is very important. The header starts it, and the footer finishes it. Everything in between will be handled by a Perl CGI script. Constructing the header and footer in this way allows the individual, page-specific Perl CGI scripts and HTML::Template files not to have to deal with the header or footer HTML content at all.

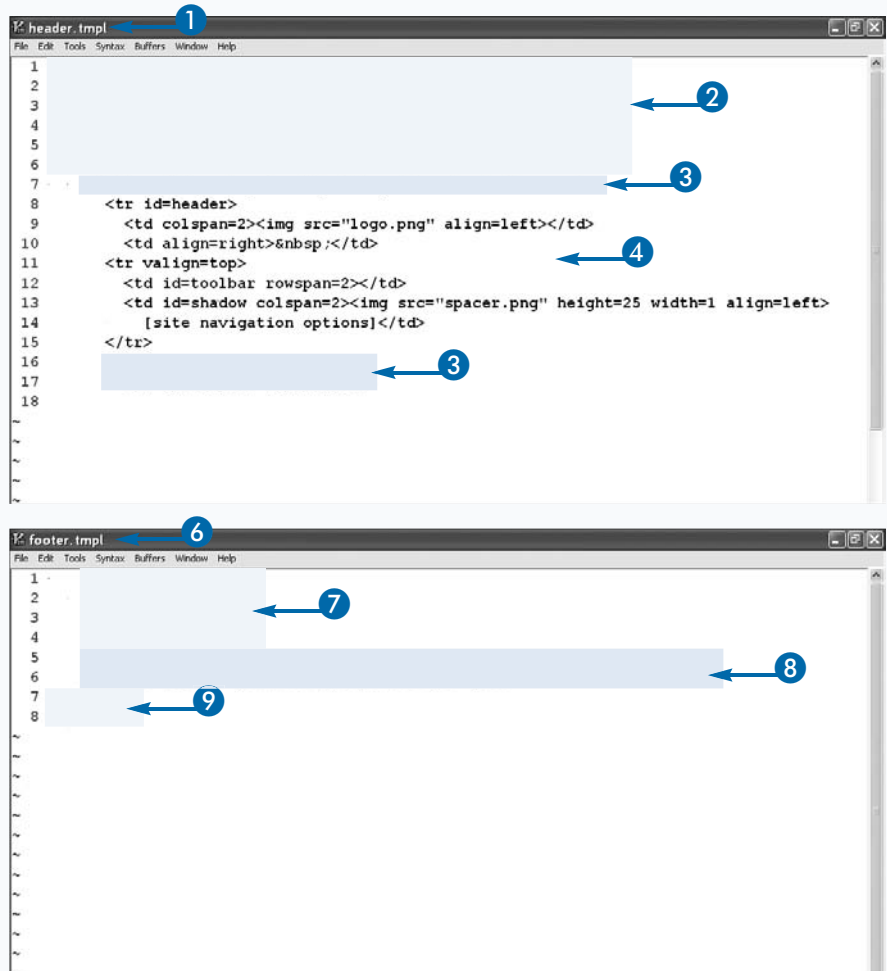
The only real point that requires special consideration is the `<title>` tag in the header. Naturally, every Web page should have its own unique title, which means that the `<title>` tag cannot be persistent.

Create an HTML::Template Header and Footer

- 1 Create the header template file in a text editor.
- 2 Insert the introductory HTML, script, style, and body tags to begin each Web page.
- 3 Begin the page-wide global table.
- 4 Include any text or images that should be at the top of every Web page.

Note: You may need to place this content strategically within the global table. You need to pre-plan how this will look best on the site.

- 5 Save the file as `header.tmpl` and close it.
- 6 Create the footer template file in a text editor.
- 7 Close the page-wide global table.
- 8 Include any text that should be at the bottom of every Web page.
- 9 Type the closing body and HTML tags to finish the Web page.
- 10 Save the file as `footer.tmpl` and close it.



Create an HTML::Template Toolbar

You can create a toolbar template for use in HTML::Template by using `TMPL_INCLUDE` within your header template. This imports the toolbar template everywhere you use the header. This allows you to standardize content across an entire Web site out of a single `header.tpl` file.

Generally speaking, in the case of a toolbar, the goal is to have some sort of logic to identify where it is on the site, and display a visual cue to the user that *you are here*. For example, on a site that has five pages — welcome, about, catalog, support, and contact — if the user is browsing the catalog page, then you want the catalog icon on the toolbar to be highlighted.

The Perl script needs to establish a specific template parameter that the toolbar template will use to indicate which page is actually selected.

Advanced toolbars may grow and shrink dynamically based upon the page that is highlighted. So, if your product catalog has a series of subcategories, then you want them to be displayed in the toolbar while on the catalog page, but hidden while on any other page.

The actual toolbar does not need to be complicated. It can be as simple as a `...` list, or as complicated as a complete `<table>...</table>`. The whole point is that it has the appropriate `TMPL_IF` statements that alter the display to clearly identify which page the user is on.

Create an HTML::Template Toolbar

- 1 Open the template file `header.tpl` in a text editor.
 - 2 Type `<tmpl_include name="toolbar.tpl">` where the new toolbar should be imported.
 - 3 Save the file as `header.tpl` and close it.
 - 4 Open a new template file in a text editor. This will represent the toolbar template.
 - 5 Write the toolbar HTML with all top-level pages available described as links.
 - 6 Type `<tmpl_if name=pagekey>class=selected</tmpl_if>` within each page link.
- Note:** The `class` statement is an example. The point is that you change the appearance of the page link when `pagekey` is true.
- 7 Apply similar changes for all page links; assign a different `pagekey` for each link.
 - 8 Save the file as `toolbar.tpl`.

```

1 <html>
2 <head>
3 <link rel="stylesheet" href="styles.css" type="text/css">
4 <title></title>
5 <body>
6 <table cellspacing=0 cellpadding=0 border=1 class=main>
7 <tr id=header>
8 <td colspan=2></td>
9 <td align=right>&nbsp;</td>
10 <tr valign=top>
11 <td id=toolbar rowspan=2></td>
12 <td id=shadow colspan=2>
13 [site navigation options]</td>
14 </tr>
15 <tr valign=top>
16 <td id=content width=100%>

```

```

1 <table id=toolbar>
2 <tr>
3 <td>
4
5
6
7
8
9
10 </td>
11 </tr>
12 <tr>
13 <td id=shadow>&nbsp;</td>
14 </tr>
15 </table>

```

```

1 <table id=toolbar>
2 <tr>
3 <td>
4 <a href="index.pl">Home</a>
5 <a <tmpl_if name=pagekey>class=selected</tmpl_if> href="about.pl">About Us</a>
6 <a <tmpl_if name=pagekey>class=selected</tmpl_if> href="catalog.pl">Catalog</a>
7 <a <tmpl_if name=pagekey>class=selected</tmpl_if> href="ordering.pl">Order in
8 <a <tmpl_if name=pagekey>class=selected</tmpl_if> href="support.pl">Support</a>
9 <a <tmpl_if name=pagekey>class=selected</tmpl_if> href="contact.pl">Contact Us</a>
10 </td>
11 </tr>
12 <tr>
13 <td id=shadow>&nbsp;</td>
14 </tr>
15 </table>

```

In other words, the user opens the Perl script directly in their browser's URL — for example, `catalog.pl`. The `catalog` Perl script initializes the `HTML::Template` module. The module loads the template file `catalog.tmpl`. The template file uses `HTML_INCLUDE` to bring in `header.tmpl`, then displays the content specific to the CGI page, and finally includes `footer.tmpl`.

The primary benefit of this technique is that the Perl script does not need to worry about shared HTML in the header, toolbar, or footer; or even the unique HTML its template file. The Perl script is dedicated to the logic in the Web page, and loading the correct master template.

Once you have implemented this method across multiple pages and CGI scripts, you can make a change to any one of the header, toolbar, or footer templates and the change will be automatically updated on every Perl CGI page — very convenient!

- 1 Create the *page1* template file in a text editor.
- 2 Type **<tmpl_include name="header.tmpl">** to include the header template.
Note: Remember, the *header.tmpl* file includes *toolbar.tmpl*.
- 3 Type some unique content, specific to this page.
- 4 Type **<tmpl_include name="footer.tmpl">** to include the footer template.
- 5 Save the template file as *page1.tmpl*.

Note: Remember, the `header.tmpl` file includes `toolbar.tmpl`.

3 Type some unique content, specific to this page.

- 4 Type `<tmpl_include name="footer.tmpl">` to include the footer template.

- 5 Save the template file as `page1.tmp1`.

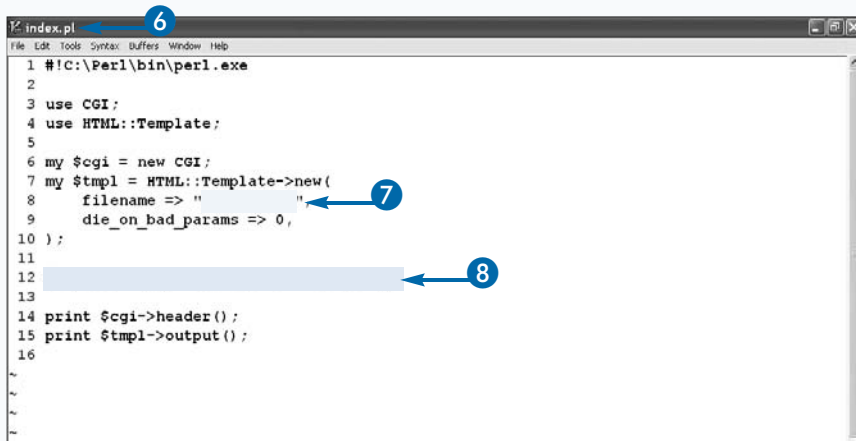
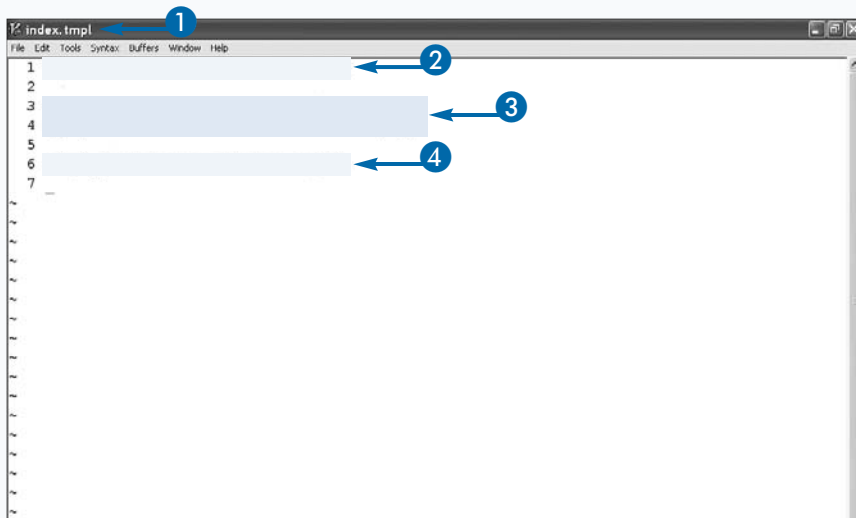
- 6 Create the *page1* Perl script in a text editor.

- 7 Make sure it uses the `page1.tmp1` template.

8 Type `$tmpl->param(pagekey1 => "active");` to activate the toolbar's highlight for *pagekey1*.

9 Save the Perl script as `page1.pl`.

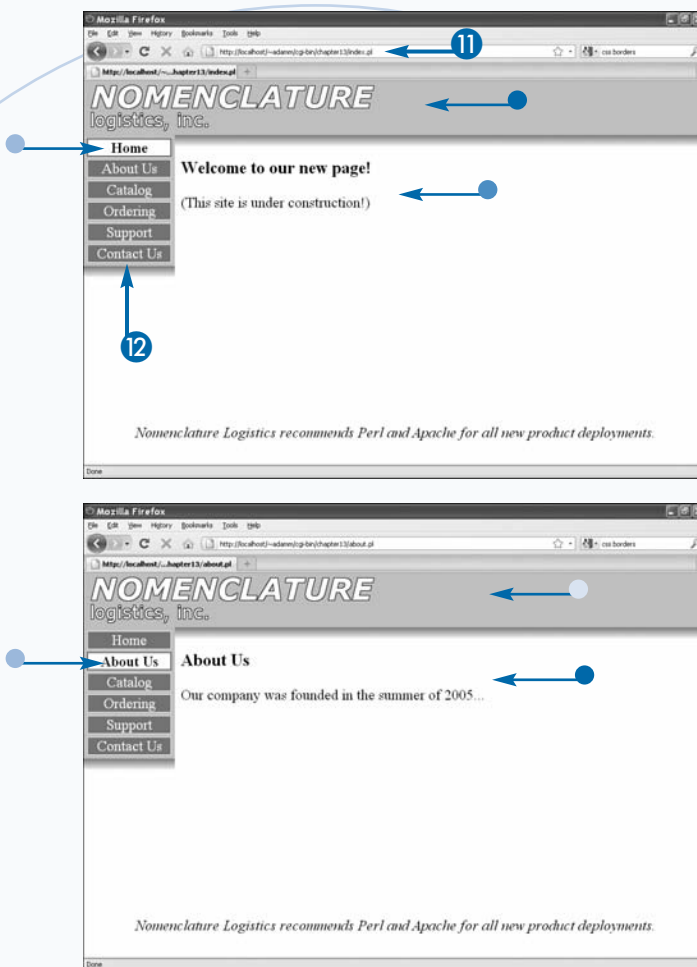
- Repeat steps 1 to 9 for `page2`. `tmpl` and `page2.pl`, and use the `pagekey2` toolbar for the highlight key.



- 11 Open *page1.pl* in a Web browser.
- The browser displays the header, toolbar, and footer.
- The browser displays the content specific to *page1*.
- The toolbar has the *page1* link highlighted.
- 12 Click the *page2* link.

The browser loads the *page2* Web page.

- The browser displays the same header, toolbar, and footer.
- The browser displays the content specific to *page2*.
- The toolbar has the *page2* link highlighted.



Apply It

All template files that you use to generate the page honor the template parameters defined by Perl. This may mean that there are other individual elements within the header or footer templates that you want to customize. The most common example of this is the Web page's title.

TYPE THIS INTO THE HEADER TEMPLATE FILE

```
<title><tmpl_var name=title
  default="Nomenclature Logistics, Inc.">
</title>
```



TYPE THIS INTO EACH PERL CGI SCRIPT

```
$tmpl->param( title => "NML: Page1 Title" );
```



RESULTS

The Web page title is customized for each page whose CGI defines a specific title; otherwise, a generic title is used.

The technique described here is specific to Web pages that have a dedicated Perl CGI script that provides content. You may find that means every page does not necessarily need a dedicated CGI script. If you have a static HTML page that you want to wrap in the header, toolbar, and footer, see Chapter 14 for more information.

Extend HTML::Template to Non-HTML Formats

You can extend an HTML::Template Perl script and template file to produce non-HTML formatted files. The actual use of this technique is rather limited when using Perl as a CGI program, as the CGI's output is typically HTML; however, you can still use Perl and HTML::Template on the command-line to produce text files that output any type of text-based format.

The obvious non-HTML candidate formats would be those that are already very close to HTML with respect to output, such as XML and XHTML. However, theoretically any ASCII-based format will work. You could even configure HTML::Template to output more Perl source code.

All of this is assuming that you can transliterate the targeted format's output into something that is communicable as a TMPL file. You still follow all the rules

stated in this chapter, which include the same template tags such as `<tmpl_var>`, `<tmpl_if>`, `<tmpl_loop>`, and `<tmpl_include>` — all are still valid. The only difference is where you would put HTML; you use whatever text-based formatting is required.

However, remember that in a Perl CGI script, the compiled HTML::Template output is simply printed onto standard-output. If you use Perl on the command-line, it is a good idea to write an actual file, with the correct extension.

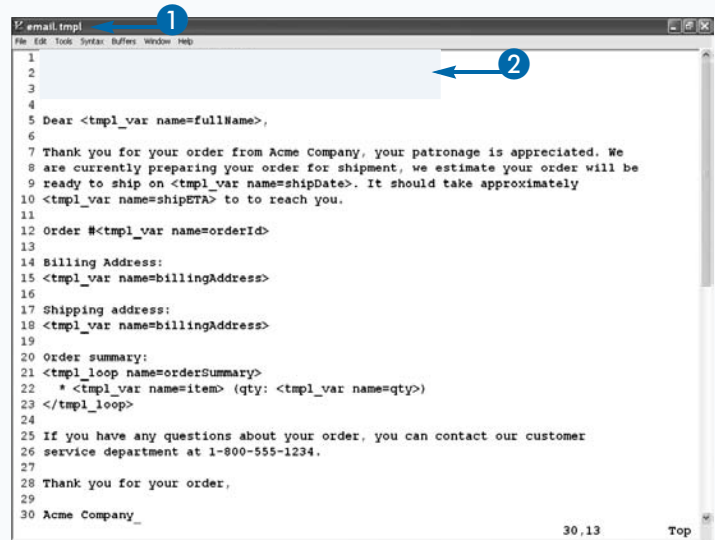
Hacking HTML::Template to produce a binary-specific file format is not recommended. For example, if you want to produce a Word or Excel document using this template system, do not even attempt to produce a DOCX or XLSX file. Instead, create a simpler TXT or CSV file and then import that into Word or Excel.

Extend HTML::Template to Non-HTML Formats

- 1 Create a new template file in a text editor.
- 2 Format the template like an e-mail message.

You can use `TMPL_VAR`, `TMPL_LOOP`, and any other HTML::Template tags.

- 3 Save the file as `email.tmpl`.

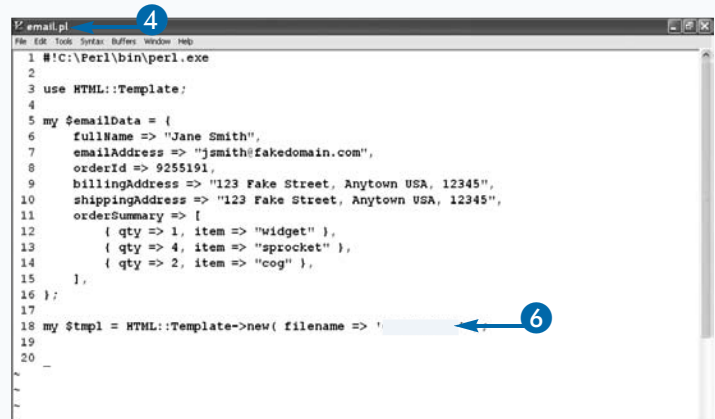


```
1
2
3
4
5 Dear <tmpl_var name=fullName>,
6
7 Thank you for your order from Acme Company, your patronage is appreciated. We
8 are currently preparing your order for shipment, we estimate your order will be
9 ready to ship on <tmpl_var name=shipDate>. It should take approximately
10 <tmpl_var name=shipETA> to reach you.
11
12 Order #<tmpl_var name=orderId>
13
14 Billing Address:
15 <tmpl_var name=billingAddress>
16
17 Shipping address:
18 <tmpl_var name=billingAddress>
19
20 Order summary:
21 <tmpl_loop name=orderSummary>
22 * <tmpl_var name=item> (qty: <tmpl_var name=qty>)
23 </tmpl_loop>
24
25 If you have any questions about your order, you can contact our customer
26 service department at 1-800-555-1234.
27
28 Thank you for your order,
29
30 Acme Company_
```

- 4 Open a Perl script that uses the HTML::Template module.
- 5 Import the data source, or insert the raw data that will be used for each loop in the template.

Note: Here you define your e-mail data by creating the completed hash ref manually. Normally, this information would be populated from an external source, such as a database.

- 6 Use `email.tmpl` as the input template filename.



```
1 #!C:\Perl\bin\perl.exe
2
3 use HTML::Template;
4
5 my $emailData = {
6   fullName => "Jane Smith",
7   emailAddress => "jsmith@fakedomain.com",
8   orderId => 9255191,
9   billingAddress => "123 Fake Street, Anytown USA, 12345",
10  shippingAddress => "123 Fake Street, Anytown USA, 12345",
11  orderSummary => [
12    { qty => 1, item => "widget" },
13    { qty => 4, item => "sprocket" },
14    { qty => 2, item => "cog" },
15  ],
16 };
17
18 my $tmpl = HTML::Template->new( filename => "
```


- 7 Type `$tmpl->param(HASHREF);` to populate each field in the e-mail message directly from the data hashref variable.

Note: This technique of plugging a single variable into `param` is only viable because all of `$emailData`'s keys already match the template's keys.

- 8 Type `print $tmpl->output();` to print the e-mail message.

- 9 Save the Perl script.

- 10 Execute the Perl script on the command-line.

- The command-line displays the completed e-mail message.

Note: At this stage you can write the output to disk, or actually send it as an e-mail message. See Chapter 12 for an example.

```

1 #!C:\Perl\bin\perl.exe
2
3 use HTML::Template;
4
5 my $emailData = {
6   fullName => 'Jane Smith',
7   emailAddress => 'jsmith@fakedomain.com',
8   orderId => 9255191,
9   billingAddress => '123 Fake Street, Anytown USA, 12345',
10  shippingAddress => '123 Fake Street, Anytown USA, 12345',
11  shipDate => 'Monday, April 1, 2010',
12  shipETA => 'four to six weeks',
13  orderSummary => {
14    { qty => 1, item => 'widget' },
15    { qty => 4, item => 'sprocket' },
16    { qty => 2, item => 'cog' },
17  },
18 };
19
20 my $tmpl = HTML::Template->new( filename => 'email.tmpl', die_on_bad_params => 0 );
21
22
23
24
25

```

```

C:\Documents and Settings\adamm\My Documents>
From: sales@acmecompany.com
To: jsmith@fakedomain.com
Subject: Your order summary #9255191

Dear Jane Smith,

Thank you for your order from Acme Company, your patronage is appreciated. We
are currently preparing your order for shipment, we estimate your order will be
ready to ship on Monday, April 1, 2010. It should take approximately
four to six weeks to reach you.

Order #9255191

Billing Address:
123 Fake Street, Anytown USA, 12345

Shipping address:
123 Fake Street, Anytown USA, 12345

Order summary:
  * widget (qty: 1)
  * sprocket (qty: 4)
  * cog (qty: 2)

If you have any questions about your order, you can contact our customer
service department at 1-800-555-1234.

Thank you for your order.

Acme Company

C:\Documents and Settings\adamm\My Documents>

```

Apply It

Your raw text file may look a little odd, in part, because everything between `<tmpl_loop>...</tmpl_loop>` is duplicated, including any new lines. One way to address this problem is to use the opening and closing `TMPL_LOOP` tags on one line.

TYPE THIS INTO THE TEMPLATE

```

Your order summary:
<tmpl_loop name=orderSummary> * <tmpl_var name=item> (qty: <tmpl_var name=qty>)
</tmpl_loop>

```



RESULTS

For every loop of `orderSummary`, an asterisk and the item's text is written, followed by a new line. This is because the `TMPL_LOOP` ends on the next line, thus including the new line character at the end of the `TMPL_VAR` text within each loop.

```

Your order summary:
* widget (qty: 1)
* sprocket (qty: 4)
* cogs (qty: 4)

```

Introducing Server-Side Includes

You can use Server-Side Includes, or SSI, to import dynamic content directly into an HTML file and to run basic conditional tests on the CGI environment. Apache understands SSI-specific elements and applies the changes to the HTML content prior to delivering the page to the Web browser.

SSI uses a new set of directives that look like HTML comments. These directives, when found in HTML,

convert into dynamic output, depending on the command used. You can use these directives to supply variables, include other HTML files, or even execute programs such as CGI scripts.

You can mix the SSI and CGI technologies, but only in a specific order. Apache cannot parse CGI output for SSI tags, but it can parse HTML files for SSI directives that reference CGI scripts.

Enabling SSI

There are two steps required to enable SSI in Apache. The first step configures a specific directory or virtual domain in Apache, which should support SSI. The second step tells Apache which files in that directory should be parsed.

Enable the SSI Module

Apache ships with a module, `mod_include.so`, which provides the SSI functionality. You must activate it before any SSI parsing will work. To do this, you need to add the following directive into the Apache configuration file:

```
LoadModule include_module modules/mod_include.so
```

The exact path to `mod_include.so` may be different on your system.

Define an SSI-Enabled Server Directory

You must enable the actual SSI capabilities in Apache by applying a new argument to the `Options` directive: `Includes`. You can apply the `Options` directive to an Apache server in multiple ways. You can set it in a specific `<directory>` or `<virtualhost>` configuration section in the Apache configuration file, or within a special file called `.htaccess` in the HTML directory. In fact, you can apply `Includes` in the same place where you enabled the CGI handler with the `ExecCGI` argument. For an example, see Chapter 10.

Define an SSI-Enabled HTML File

There are two ways to instruct Apache to parse HTML files for SSI directives. You can use either method, but you only need one.

The first method involves renaming the HTML file with the extension `.shtml`; you then add two new Apache configuration directives:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

However, renaming an existing `.html` file is not always practical, as you also need to update all links to that page.

The second method is to enable the `XBitHack` in Apache. This allows you to simply set the execute-bit on an HTML file, just like you would in a Perl CGI script, and Apache processes its output for SSI code:

```
XBitHack on
```

Unfortunately, the `XBitHack` is only available on Unix installations of Apache. Windows has no concept of an execute-bit, so you must use the file rename method.

Using SSI

Whether you use SSI depends on your personal preference and experience. If you already have static HTML files that need to be extended to support dynamic content, SSI is the best option. If you do not have any static HTML files, and everything is handled by the Perl CGI interface, then you do not need SSI. However, you can layer SSI and CGI to enable both technologies.

SSI Elements

You format all SSI elements in HTML using the following syntax:

```
<!--#element attribute="value" ... -->
```

Spacing is very specific here. There must be a space between the `element` and `attribute` keywords, a space between each `attribute="value"` pair, and finally a space preceding the closing `-->` tag.

The individual elements available are described in the section, "Understanding SSI Elements."

Layering SSI, Perl CGI, and HTML::Template

Choosing to layer SSI, Perl CGI, and HTML::Template on top of one another involves some discipline in your code, and an understanding of what each layer does, but the advantage of doing so is that you produce a very powerful Web site.

Layering involves starting with a static HTML file that contains the source content for a page. This could be an *About* or *Contact Us* page that contains a few paragraphs that rarely change. This HTML file is the Web page URL.

Even though the actual page does not change, it may reference dynamic content that does. This could include other features on the Web site that are more dynamic, such as the site header or a navigational toolbar. So, the *About* HTML page file contains SSI elements that include separate Perl CGI scripts.

The included header or toolbar CGI may also react dynamically to the user. For example, the header may contain a summary of the user's current shopping cart, or the navigation toolbar may list the other products or pages the user has recently visited. These CGI scripts would then outsource their layout to HTML::Template.

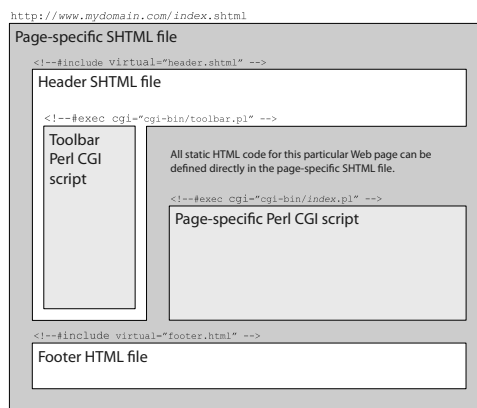
Dangers of SSI

You can use the `#exec` element to execute a program on the server and forward its output to the browser. For example, on Unix you can use the `uptime` program to display the last time the server was rebooted. If you want to provide this feature on your Web site somewhere, use this element:

```
<!--#exec cmd="uptime" -->
```

The danger with enabling this particular feature becomes evident if you allow users to enter data in a form, which in turn is displayed as HTML back to the browser. Imagine what would happen if the user entered in a raw SSI command in a form such as `<!--#exec cmd="cat / etc/passwd" -->` and then viewed it as HTML? The user could execute any command on the server from their Web browser!

To disable the `#exec` SSI element, activate SSI with the `IncludesNoExec` argument, not `Includes`, within the `Options` directive.



Enable the Apache SSI Module and Output Filter

Enabling an SSI Handler in Apache allows you to parse HTML output for SSI elements in a browser. To do this, two things need to happen: first you must enable the SSI module, and then you must enable the output filter.

You do this by enabling two new Apache directives. The first directive instructs Apache to deliver files with the .shtml extension as the text/html MIME-type; the second forwards all .shtml files through the SSI output-filter:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

You enable the module and the output filter in the Apache configuration file. The exact configuration file depends on your operating system. If you are using

Windows, edit `Apache Install Dir\conf\httpd.conf`. If you are on Debian- or Ubuntu-based Linux, edit `/etc/apache2/mods-enabled/mime.conf`. For Red Hat-based Linux, edit `/etc/httpd/conf/httpd.conf`.

Because this is a two-step process, on Windows and Red Hat-based systems (including Mandriva and Fedora), you only need to change one configuration file. If you are using Debian or Ubuntu Linux, you can enable the SSI module using the command, `a2enmod`, and then configure the SSI handler in the `mime.conf` file.

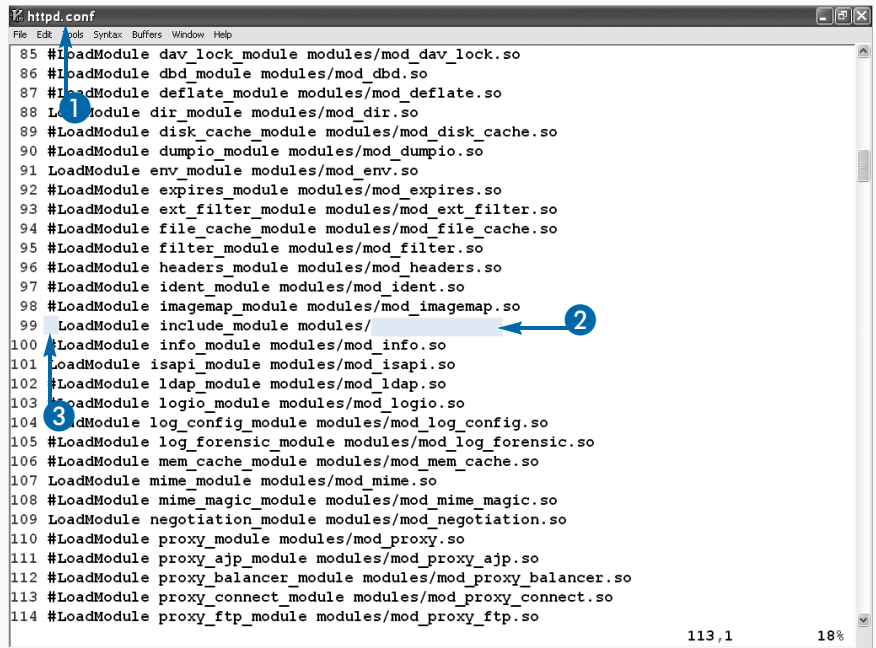
Once the SSI module and output filter are online, you need to assign a directory to the module. For more information, see the section, “Configure a Directory to Use SSI.” The final step is to rename all SSI-aware files from `filename.html` to `filename.shtml`.

Enable the Apache SSI Module and Output Filter

- 1 Open the Apache configuration file containing the SSI module directives in a text editor.

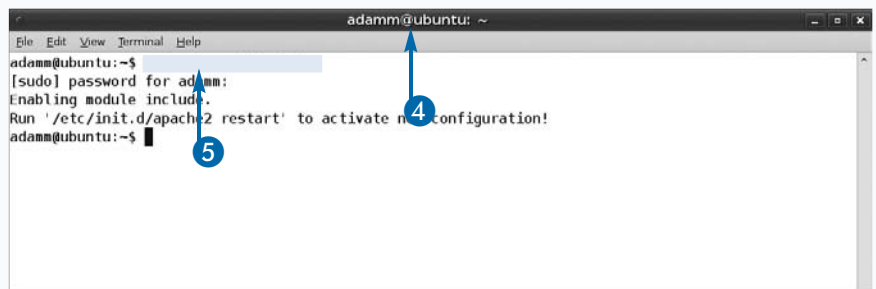
Note: If you are using Debian, Ubuntu, or Mandriva Linux, go to step 4.

- 2 Search the file for the directive `LoadModule include_module`.
- 3 Remove the hash (#) to uncomment the directive, if it is commented out.
- 4 Open a Terminal window.
- 5 (Debian and Ubuntu Linux only) Type **`sudo a2enmod include`** and press Enter.
(Mandriva Linux only) Type **`gurpmi apache-mod_include`** and press Enter.
- 6 Type your password if prompted and press Enter.



```
File Edit Tools Syntax Buffers Window Help
85 #LoadModule dav_lock_module modules/mod_dav_lock.so
86 #LoadModule dbd_module modules/mod_dbd.so
87 #LoadModule deflate_module modules/mod_deflate.so
88 #LoadModule dir_module modules/mod_dir.so
89 #LoadModule disk_cache_module modules/mod_disk_cache.so
90 #LoadModule dumpio_module modules/mod_dumpio.so
91 LoadModule env_module modules/mod_env.so
92 #LoadModule expires_module modules/mod_expires.so
93 #LoadModule ext_filter_module modules/mod_ext_filter.so
94 #LoadModule file_cache_module modules/mod_file_cache.so
95 #LoadModule filter_module modules/mod_filter.so
96 #LoadModule headers_module modules/mod_headers.so
97 #LoadModule ident_module modules/mod_ident.so
98 #LoadModule imagemap_module modules/mod_imagemap.so
99 LoadModule include_module modules/
100 #LoadModule info_module modules/mod_info.so
101 LoadModule isapi_module modules/mod_isapi.so
102 #LoadModule ldap_module modules/mod_ldap.so
103 #LoadModule logio_module modules/mod_logio.so
104 #LoadModule log_config_module modules/mod_log_config.so
105 #LoadModule log_forensic_module modules/mod_log_forensic.so
106 #LoadModule mem_cache_module modules/mod_mem_cache.so
107 LoadModule mime_module modules/mod_mime.so
108 #LoadModule mime_magic_module modules/mod_mime_magic.so
109 LoadModule negotiation_module modules/mod_negotiation.so
110 #LoadModule proxy_module modules/mod_proxy.so
111 #LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
112 #LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
113 #LoadModule proxy_connect_module modules/mod_proxy_connect.so
114 #LoadModule proxy_ftp_module modules/mod_proxy_ftp.so

113,1 18%
```



```
adamm@ubuntu: ~
File Edit View Terminal Help
adamm@ubuntu:~$ sudo a2enmod include
[sudo] password for adamm:
Enabling module include.
Run '/etc/init.d/apache2 restart' to activate new configuration!
adamm@ubuntu:~$
```

- 7 Search the file for the directives that enable SSI for .shtml files.
- 8 Remove the hash (#) to uncomment both directives, if they are commented out.
- 9 Save the Apache configuration file.

```

385 # actions unrelated to filetype. These can be either built into the server
386 # or added with the Action directive (see below)
387 #
388 # To use CGI scripts outside of ScriptAliased directories:
389 # (You will also need to add "ExecCGI" to the "Options" directive.)
390 #
391 AddHandler cgi-script .cgi .pl
392
393 # For type maps (negotiated resources):
394 #AddHandler type-map var
395
396 #
397 # filters allow you to process content before it is sent to the client.
398 #
399 # To parse .shtml files for server-side includes (SSI):
400 # (You will also need to add "Includes" to the "Options" directive.)
401 #
402 AddType text/html
403 AddOutputFilter INCLUDES
404 </IfModule>
405
406 #
407 # The mod_mime_magic module allows the server to use various hints from the
408 # contents of the file itself to determine its type. The MIMEMagicFile
409 # directive tells the module where the hint definitions are located.
410 #
411 #MIMEMagicFile conf/magic
412
413 #
414 # Customizable error responses come in three flavors:

```

- 10 Type **net stop apache2.2** to stop the Apache service.
- 11 Type **net start apache2.2** to restart the Apache service.

Note: The **net stop** and **net start** commands are specific to Windows as a shortcut to the Services control panel. To start and stop the Apache Service on Linux, see Chapter 5.

```

C:\Program Files\Apache Software Foundation\Apache2.2\conf>gvim httpd.conf
C:\Program Files\Apache Software Foundation\Apache2.2\conf>
The Apache2.2 service is stopping.
The Apache2.2 service was stopped successfully.

C:\Program Files\Apache Software Foundation\Apache2.2\conf>
The Apache2.2 service is starting.
The Apache2.2 service was started successfully.

C:\Program Files\Apache Software Foundation\Apache2.2\conf>

```

Extra

Normally, after SSI is fully configured, files with the extension .shtml are parsed for SSI elements. However, this is not the only way to activate SSI in Apache. When using Apache on Unix, you can enable a directive called an XBitHack to simplify this setup process. Once active, you only need to set the execute-bit on an individual HTML file for it to be parsed for SSI elements, regardless of the file extension used. Open the Apache configuration file and add the following directive:

```
XBitHack on
```

Once restarted, set all HTML files you want parsed for SSI elements with this command:

```
chmod +x filename.html
```

Alternatively, you can instruct Apache to parse *all* HTML files for SSI elements, eliminating the need to rename the file to filename.shtml, or to use the XBitHack directive. However, if you employ this method, Apache parses every HTML file for SSI elements that it serves. This may affect the performance of your Web server. To do this, use the following directive in the Apache configuration file:

```
AddOutputFilter INCLUDES .html
```

Configure a Directory to Use SSI

Once you have enabled the SSI module, the final step is to configure a directory to be parsed for SSI tags. Unlike when you enabled the CGI module under a special `cgi-bin` directory, you do not require a dedicated directory. This allows you to create Web page files that will be parsed by Apache for SSI elements when requested by a Web browser.

You need to open the Apache configuration file and edit the `<Directory>` configuration section; locate its `AllowOverride` directive, and append the value `Options`. The location of the Apache configuration file depends on your operating system. If you are using Windows, edit `Apache Install Dir\conf\httpd.conf`. If you are on Debian- or Ubuntu-based Linux, edit `/etc/apache2/mods-enabled/userdir.conf`. For

Mandriva Linux, edit `/etc/httpd/modules.d/67_mod_userdir.conf`. For all other Red Hat-based Linux distributions, edit `/etc/httpd/conf/httpd.conf`. Within the directory that will house SSI-friendly HTML files, create a special file called `.htaccess` with exactly one line:

```
Options +Includes
```

This grants full SSI rights, including program-execution capabilities, to the HTML files in this directory. Execution permissions can be a security risk, especially if you use a shared Web server. To mitigate this risk, enable SSI without execution rights using this command instead:

```
Options +IncludesNoExec
```

After setting the `AllowOverride` directive and creating the `.htaccess` file, simply rename your HTML files to have an `.shtml` extension. Alternatively, if you used the `XBitHack` method on Unix, you do not have to alter the extension, but you need to enable the execute-bit on the actual HTML file.

Configure a Directory to use SSI

- 1 Open the Apache UserDir configuration file in a text editor.
- 2 Locate the `<Directory>` configuration section for UserDir directories.
- 3 Type **Options** as a new value in the `AllowOverride` directive.
- 4 Save the UserDir configuration file and exit the editor.
- 5 Restart Apache.
- 6 Go to the `My Website` directory (`public_html` in Linux) and open the file `.htaccess`.

Note: Create the `.htaccess` file if it does not yet exist.

- 7 Type **Options +Includes** to enable SSI in this directory.
- 8 Save the `.htaccess` file.

Note: Directives written to `.htaccess` files do not require Apache to be restarted to implement their functionality.

```
1 # Settings for user home directories
2 #
3 # Required module: mod_userdir
4 #
5 #
6 # UserDir: The name of the directory that is appended onto a user's home
7 # directory if a ~user request is received. Note that you must also set
8 # the default access control for these directories, as in the example below.
9 #
10 UserDir "My Documents/My Website"
11 #
12 #
13 # Control access to UserDir directories. The following is an example
14 # for a site where these directories are restricted to read-only.
15 #
16
17 AllowOverride FileInfo AuthConfig Limit Indexes
18 Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
19 <Limit GET POST OPTIONS>
20     Order allow,deny
21     Allow from all
22 </Limit>
23 <LimitExcept GET POST OPTIONS>
24     Order deny,allow
25     Deny from all
26 </LimitExcept>
27
28
```

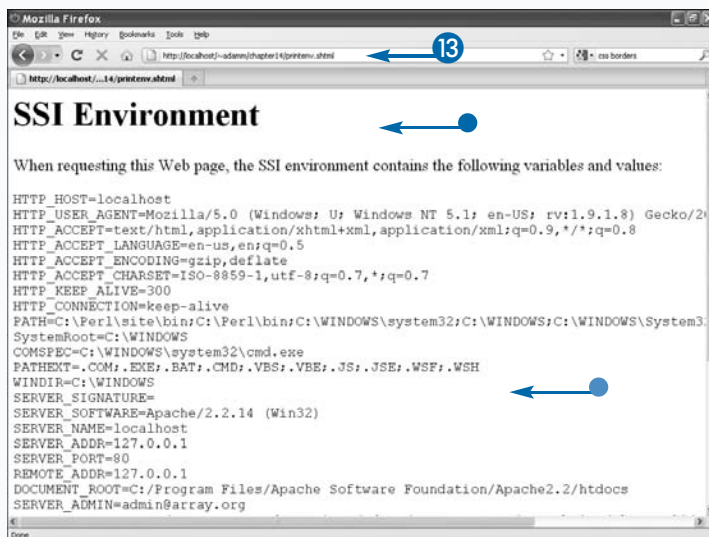
```
1 Options +Includes
```


- 9 Create a new SHTML file.
 - 10 Type some regular HTML code.
 - 11 Type `<!--#printenv -->` to print all environment variables from SSI.
 - 12 Save the SHTML file as `filename.shtml`.
- (Unix-only) If you used the `XBitHack` method, type the command **`chmod +x filename.shtml`** to activate SSI.
- 13 In a browser, go to the SHTML file's URL.
 - The browser displays the static HTML in the SHTML file.
 - The browser displays the SSI-parsed output of the SHTML file.

```

1 <html>
2 <body>
3 <h1>SSI Environment</h1>
4 <!--#printenv -->
5 <p>When requesting this Web page, the SSI environment contains the following
6   variables and values:
7 </pre>
8 </body>
9 </html>

```



Extra

After you complete this step, you are ready to start applying some SSI elements to your HTML code. When testing, you may find that SSI does not appear to be working correctly; one of the steps required may be incomplete or incorrect. Remember, you need to enable the Apache module, activate the Apache output filter, assign a directory that Apache will watch for SSI content, and finally activate the SSI parser on individual HTML files.

This may seem like a lot of steps, but you only need to complete the first three when enabling SSI in Apache. You only need to complete the final step, activating the SSI parser on an individual file, when you create a new SSI-compliant file.

If you do not properly enable each SSI component, Apache will silently ignore the SSI-specific functionality. In fact, if you view your Web page's source code, you will notice raw SSI tags showing up as literal HTML comments.

Be sure to consult with the Apache error logs. They should be able to provide you with a hint as to where the problem is.

Understanding SSI Elements

You can use SSI elements stored within your SHTML files to access environment variables, include external files, and test for basic conditional expressions. The SSI tag uses the same structure as an HTML comment: they both begin and end with `<!--` and `-->`. Like an HTML comment, the text within is not visible when viewing your Web page in a browser, but it is visible if a user views your page's source code. The SSI tag is a specific syntax that references the SSI element and any required attributes:

```
<!--#element attribute="value" ... -->
```

When Apache serves your Web page to the Internet, the SSI module parses the SHTML file looking for the characters beginning with `<!--#` and ending with `-->`. (A space must precede the closing tag boundary.) If this format is recognized, the SSI parsing mechanism is activated and the SSI element is processed. The SSI element used may require one or more `attribute="value"` pairs. You may wrap the attribute's value in double-quotes.

Basic SSI Elements

Basic SSI elements are individual SSI tags that perform a single, read-only function. This is usually as simple as retrieving and displaying server-based information within the Web site's original HTML output.

Importing Content

The most common use of SSI is to import content from external files or programs, and superimpose that content within normal HTML output. The SSI elements `include` and `exec` provide this functionality. If either element cannot find the referenced file, a generic error message appears in the HTML output.

External Files

Static text files, such as HTML, can be imported with SSI using the `include` element with either the `file` or `virtual` attributes:

```
<!--#include file="file" -->
<!--#include virtual="file" -->
```

The difference is that `file` looks for the file relative to the Web server's hard drive, and `virtual` looks for the file relative to the Web site's domain URL.

Executing CGI Scripts

CGI scripts may be executed using the `include` element, but some important CGI data is not forwarded to the script. Instead, use the `exec` element with the `cgi` attribute:

```
<!--#exec cgi="script_file" -->
```

Like `include virtual`, the actual script file is located relative to the domain root.

Executing Programs

SSI can execute a compiled binary whenever you use the `exec` element with the `cmd` attribute. The data the executable prints on standard-output is then relayed to the Web browser.

```
<!--#exec cmd="program" -->
```

Naturally, only command-line programs work with this feature. Programs with a graphical user interface, or GUI, do not function correctly as SSI reads only the data printed to standard-output.

Basic SSI Elements *(continued)*

Display Environment Variables

SSI variables are stored within the Apache session environment, which is mostly populated with CGI-specific variables. To display a complete list of all environment variables and their values, use the `printenv` element:

```
<!--#printenv -->
```

To display specific variable data in the HTML output, use the `echo` element:

```
<!--#echo var="name" -->
```

Display File Statistics

You can use SSI to display information about additional files on the Web server, such as a file's last-modified timestamp or its overall size on disk. This can describe to the user when a specific file was last changed by the Web site administrator, or it can display the total size of a file that can be downloaded.

The `fsize` and `flastmod` elements provide this functionality. Either element takes exactly one attribute, either `file` or `virtual`. This allows you to specify the file path on the Web server, or a virtual path from the root of the domain being served.

Advanced SSI Elements

Advanced SSI elements alter the flow of the original HTML. This means that you can use them to display or omit a portion of the HTML code, in accordance with conditional testing on CGI environment variables.

Setting Variables

You can use the `set` element to set a new variable or change an existing one. Once you assign a new value, you can use it for testing with the `if` or `elif` elements, and display it with the `echo` element:

```
<!--#set var="name" value="value" -->
```

The variable name you are setting does not need to be predefined, but be careful about setting existing CGI environment variables. Doing so overrides the original value provided by Apache or the user's Web browser, which may adversely affect normal CGI operations.

Conditional Tests on Variables

Conditional SSI elements allow you to test SSI variables for specific data, and to control the output of portions of HTML based upon those results. There are four SSI elements that provide this functionality: `if`, `elif`, `else`, and `endif`. When you use them together, they form a complete *conditional test block*.

The block begins with an `if` element. This establishes an initial test expression, defined by the `expr` attribute.

Optionally, you can use `elif` and `else` to provide additional logic if the preceding `expr` returns false. Because `elif` is a shortcut for *else* and *if*, it also requires an `expr` attribute.

The block ends with the `endif` element:

```
<!--#if expr="TEST1" --> TEST1 is true!
<!--#elif expr="TEST2" --> TEST2 is true!
<!--#else --> TEST1 and TEST2 are false!
<!--#endif -->
```

The content that is written in between the elements defines what the browser will display as HTML output, but only if the preceding logic allows it.

The syntax for the `expr` attribute is fairly robust. SSI variables are referenced just like Perl scalars: `$var`. Hard-coded strings should be single-quoted so that they do not interfere with the `expr` statement's double-quotes:

```
<!--#if expr="$HTTPS != 'on'" -->Warning: SSL
is not active!<!--#endif -->
```

SSI can test using simple equality operators, and even supports nesting multiple test conditions using brackets.

Error Messages

If an SSI element uses incorrect syntax, or has any problem performing the requested command, an error message appears: *[an error occurred while processing this directive]*. You can customize this error message with `config errmsg`.

Import Files with SSI

You can import external files into your HTML output with SSI. Typical applications of this feature are to re-use common code, such as a header or footer, across multiple SHTML pages, or to import data generated by another program by way that program's standard-output (STDOUT) display data.

You can use the SSI element `include`, with either the `file` or `virtual` attributes:

```
<!--#include file="file" -->
<!--#include virtual="file" -->
```

Either attribute method works. The difference is that `file` searches for the file relative to the Web server's file-system, while `virtual` searches relative to the Web site's domain. Note that the system-user account that runs the Apache service requires read-permissions on files.

Import Files with SSI

- 1 Create a new SHTML file.
- 2 Type some regular HTML code.
- 3 Type `<!--#include virtual="filename" -->` to import this file.
- 4 Save the SHTML file.

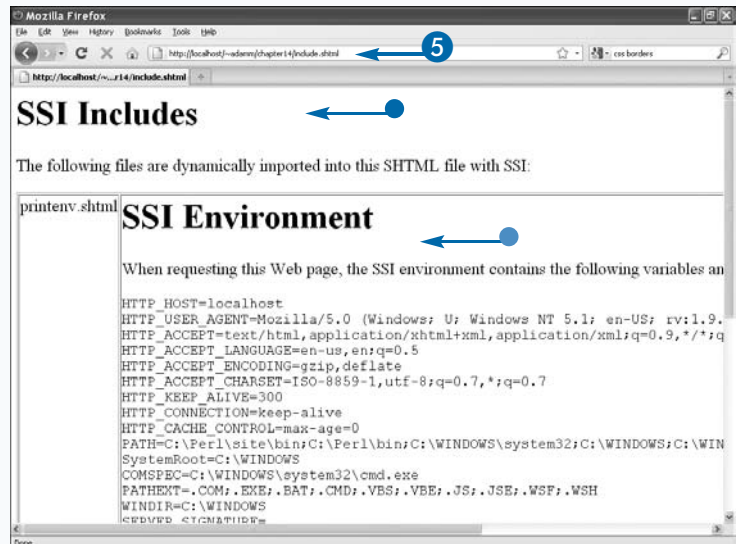
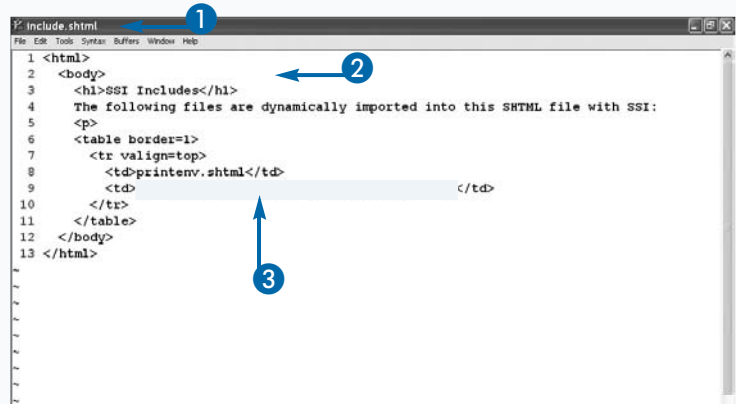
- 5 In a browser, go to the SHTML file's URL.
 - The browser displays the regular file.
 - The browser displays the imported file.

Deciding which attribute method to use may be confusing, but it all depends on where the source file exists. If it exists outside of the Apache "visible" directory path, or in other words, is not accessible by any URL, then you can use the `file` attribute. If your source file exists within the same domain URL, you can use the `virtual` attribute.

If the Apache SSI parser cannot find the file, a generic error message is displayed:

```
[an error occurred while processing this directive]
```

It is possible to execute CGI scripts with `include`; however, the CGI environment may not be forwarded to the CGI script properly, depending on your version of Apache. Instead, use `exec cgi` as described in the section, "Execute Programs with SSI."



Execute Programs with SSI

You can use the `exec` SSI element to import the run-time output of an external program into your SHTML file. Execution happens in real-time, meaning that every time a user visits a Web page with this element enabled, the program executes immediately and its output displays.

You can use either the `cmd` or `cgi` attributes to reference a binary command-line program, or a CGI script, respectively:

```
<!--#exec cmd="program_file" -->
<!--#exec cgi="script_file" -->
```

Note that there are security concerns with enabling the `exec` option. This can be very dangerous if untrusted users have the opportunity to upload HTML code to your Web site that will be processed

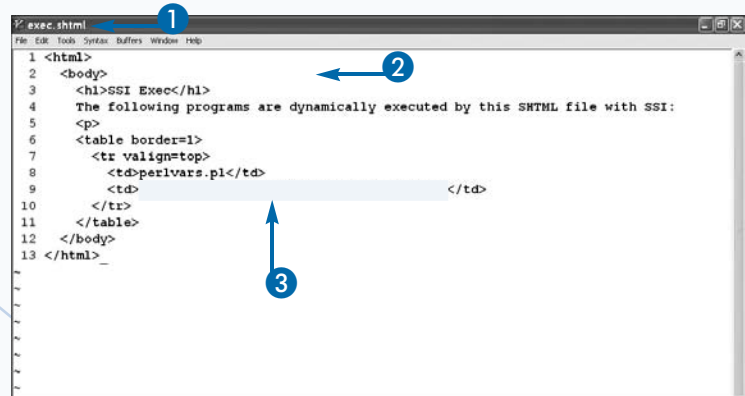
through SSI. The impact could range from viewing files outside of the Apache `htdocs` directory, to deleting files, depending on the Apache user's effective privileges on your server.

To remove this risk, you can configure Apache to provide SSI functionality without execution privileges by applying the `IncludesNoExec` option into `.htaccess`, or the Apache master configuration file. If `IncludesNoExec` is set, you cannot use `exec` at all. It is still possible to execute CGI scripts with the `include` element, but the CGI environment may not be set up correctly depending on your version of Apache. Specifically, the `QUERY_STRING` field may be absent from the CGI script's environment.

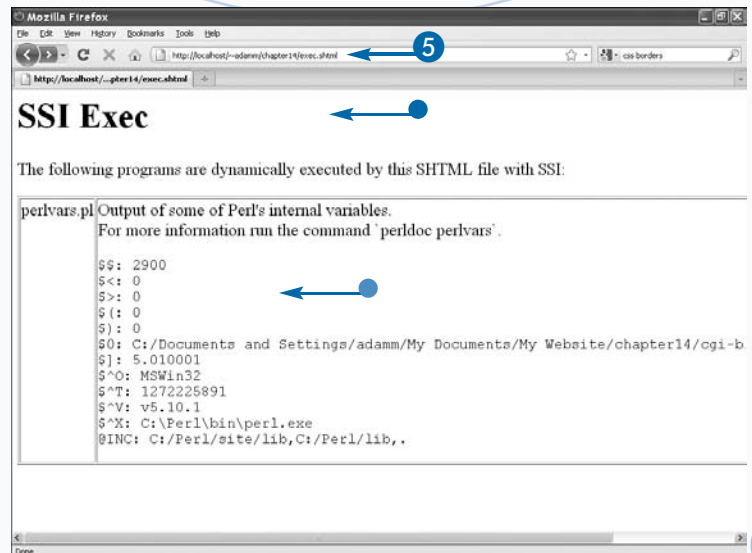
If you require HTML forms to both work through SSI and call an underlying CGI script, test both `include virtual` and `exec cgi` methods.

Execute Programs with SSI

- 1 Create a new SHTML file.
- 2 Type some regular HTML code.
- 3 Type `<!--#exec cgi="script" -->` to execute this CGI script.
- 4 Save the SHTML file.



- 5 In a browser, go to the SHTML file's URL.
 - The browser displays the regular file.
 - The browser displays the CGI program's output.



Set Variables within SSI

You can set variables within SSI to be read by other SSI elements, or by a Perl CGI script. In other words, you can assign unique data early in the SHTML Web page, and do something with it later in the SSI workflow:

```
<!--#set var="name" value="value" -->
```

Once a variable is set, its value can be retrieved using the `echo` element, or tested against with `if` and `elif` in SSI, or in an included Perl CGI script using the `%ENV` hash. By default, all of SSI's variables are inherited from the CGI environment. Be careful about setting a variable using a pre-defined CGI keyword; no warning message displays if you do this.

When the variable is assigned, or re-assigned, the value is only temporary. The change is only applicable after `set` is called, and lost when Apache finishes serving the Web

page request. In other words, their values are not persistent across multiple Web pages.

The `set` functionality is most typically used when building a Web site that imports a common HTML file, such as a toolbar, within multiple individual HTML files, such as an *Index* or *Contact Us* page. For example, you could set a new variable in `index.shtml` that assigns `"page=index"`, and in `contact.shtml` that assigns `"page=contact"`. Because both pages include `toolbar.shtml`, the toolbar's logic tests for `"page"` to identify where the user is on the Web site and highlights the toolbar accordingly. Subsequently, when `toolbar.shtml` executes `toolbar.pl`, the Perl CGI can identify the user's location using `$ENV{ 'page' }`.

A complete description and implementation of this scenario is described in the section, "Link the Header, Toolbar, and Footer with Static HTML Content."

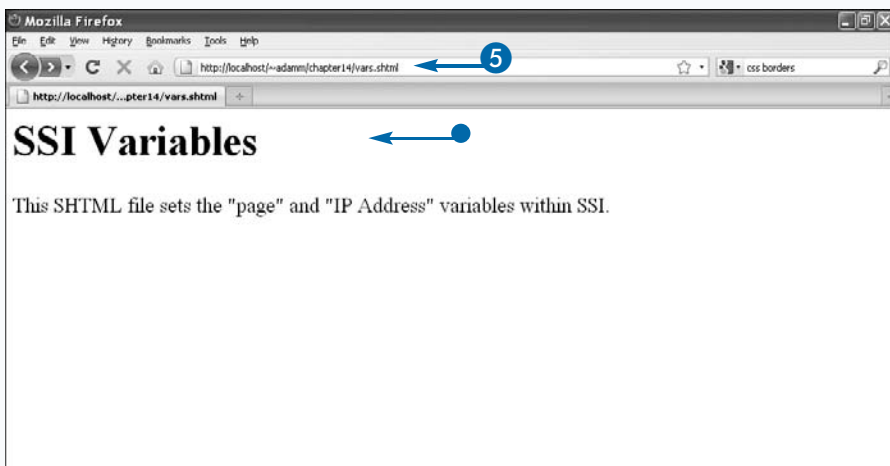
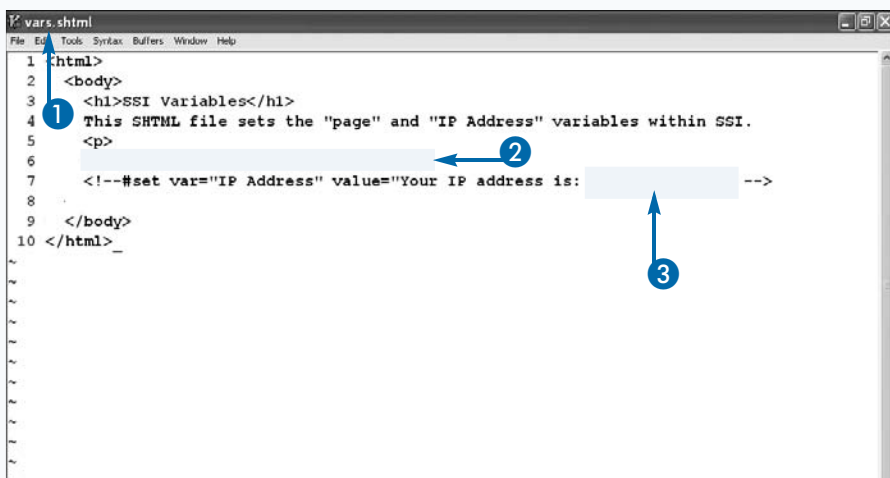
Set Variables within SSI

- 1 Open an SHTML file.
- 2 Type `<!--#set var="name" value="value" -->` to set an SSI variable.
- 3 Use `${name}` when referring to an SSI variable that is assigned into another SSI variable.
- 4 Save the SHTML file.

- 5 In a browser, go to the SHTML file's URL.
 - The browser displays the regular file.

The browser does not display additional SSI content confirming any new variables or values.

Note: SSI variables are stored within the Apache session. Nothing appears in the browser when setting them, even if you view the page's source code.



Retrieve Variables with SSI

You can access SSI variables with the `echo` element. You can then display a variable's data in the HTML output stream being sent to the user's Web browser. Along with built-in SSI variables, you can also access all CGI environment and user-defined variables:

```
<!--#echo var="name" -->
```

To see a complete list of all variables and their values, use the `printenv` element:

```
<!--#printenv -->
```

With some clever planning, you could even use `echo` to pre-populate and manipulate content outside of visible HTML. For example, you can easily populate a JavaScript variable or hidden HTML input field with a default value assigned through SSI:

```
<script type="text/javascript">
var userIP = "<!--#echo var="REMOTE_ADDR"
-->";
</script>
<input type="hidden" name="userIP"
value="<!--#echo var="REMOTE_ADDR" -->">
```

Notice the double-quotes appear incorrectly nested. It is true you should never have a pair of double-quotes that do not define a single contextual statement; however, there are two different technologies doing the parsing here. Because SSI parses the content before the Web browser, it starts at `<!--#` and stops at `-->`, so it only sees the double-quoted `var` value. By the time the browser receives the data, the SSI tag has already been converted into the user's public IP address. The JavaScript and HTML form input parsers will read their double-quoted IP address assignments correctly.

Retrieve Variables with SSI

- 1 Open an SHTML file.
- 2 Type `<!--#echo var="name" -->` to get an SSI variable's value.
- 3 Type `<pre><!--#printenv --></pre>` to display the environment.

Note: The `printenv` SSI tag includes carriage returns, but no `
` tags; `<pre>` ensures that it displays correctly.

- 4 Save the SHTML file.
- 5 In a browser, go to the SHTML file's URL.
 - The browser displays the individual variables.
 - The browser displays the complete list of variables and values.

Note: If you scroll to the bottom of `printenv`'s output, you will also find your newly created SSI variables here.

Use Conditional Expressions with SSI

SSI allows you to use conditional expressions to control what the browser displays to the user, depending on the results of a conditional expression test. You can place additional content, such as unique HTML or SSI tags, in between each conditional expression tag to be displayed when a test permits it. If a test returns true, the Web browser displays the enclosed content, but no other content from any other test:

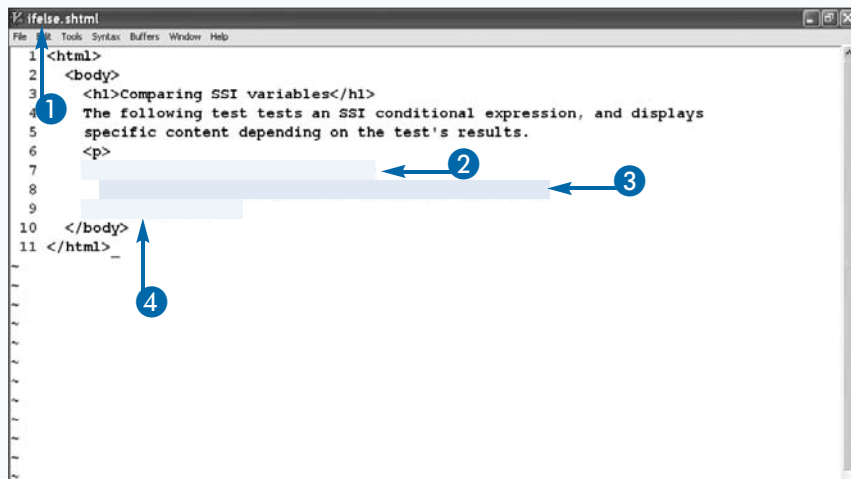
```
<!--#if expr="condition1" -->
...
<!--#elif expr="condition2" -->
...
<!--#else -->
...
<!--#endif -->
```

A conditional expression block must always begin with an `if` element and an `expr` attribute to define the test logic. If the first test fails, you can add an optional `elif` element with a new `expr` test. An optional `else` element may follow, whose content only displays if the previous `expr` logic returns false. Finally, you must use the `endif` element to complete the conditional expression block.

The formatting of `expr`'s syntax is fairly intuitive: standard equality operators such as `=` and `!=` are common. Do not use `<`, `>`, `<=`, or `>=` as SSI is only capable of comparing strings by their ASCII character values, not by their number values. You can nest tests together with brackets, and you can compare them as a group with the binary operators `&&` and `||`. You can use any SSI variable within a conditional expression test; precede the variable name with a dollar-sign, just like a Perl scalar.

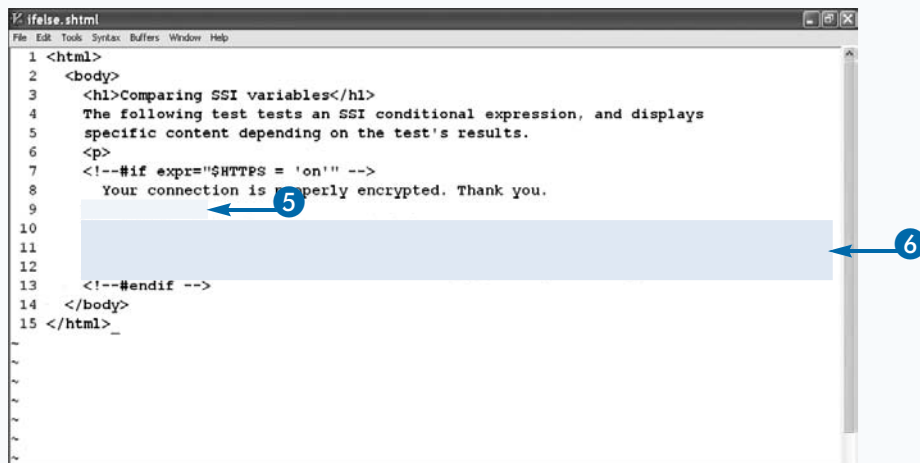
Use Conditional Expressions with SSI

- 1 Open an SHTML file.
- 2 Type `<!--#if expr="$VAR = 'value'" -->` to begin a conditional expression block.
- 3 Type some content that you want to display if the conditional statement is true.
- 4 Type `<!--#endif -->` to complete the block.



Note: At this stage, the simplest conditional expression block is complete. You have the option to include the `elseif` and `else` statements within the block to customize your display logic.

- 5 Type `<!--#else -->` after the true content, but before `<!--#endif -->`.
- 6 Type some content that you want to display if the previous conditional statement is false.



Note: You can stop here, or you can continue to fine-tune the display logic.

7 Type `<!--#elseif expr="condition" -->` before the `<!--# else -->` statement.

8 Type some content that you want to display if the previous conditional statement is false.

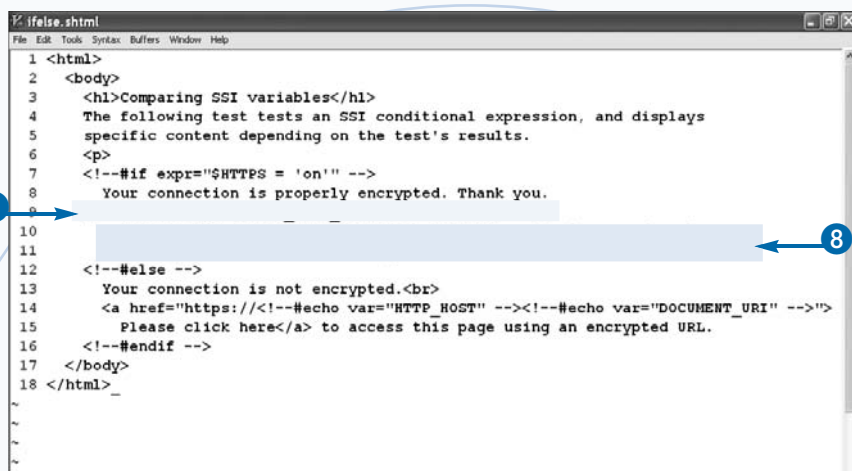
Note: You can specify multiple `elseif`s after the first `if` and before `else` or `endif`.

9 Save the SHTML file.

10 In a browser, go to the SHTML file's URL.

- The browser displays specific content based upon the various conditional tests.

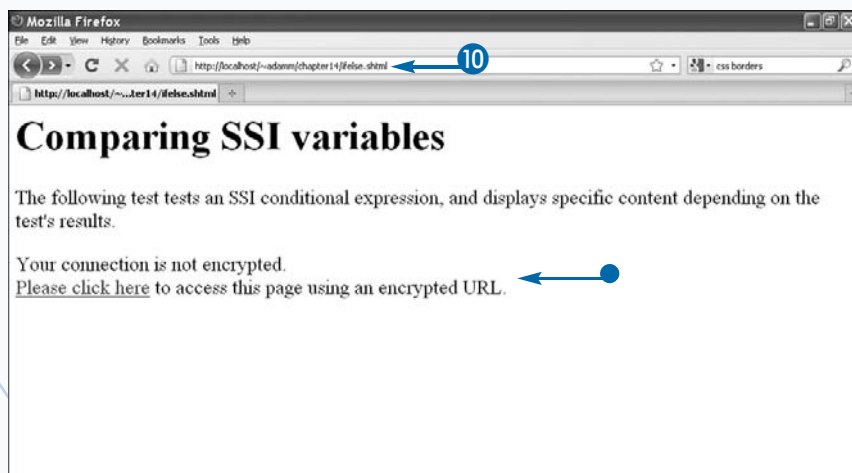
Note: If you view the source code to this page in your browser, you will see that only the "true" display section of the conditional block is sent to the browser.



```

1 <html>
2 <body>
3   <h1>Comparing SSI variables</h1>
4   The following test tests an SSI conditional expression, and displays
5   specific content depending on the test's results.
6   <p>
7     <!--#if expr="$HTTPS = 'on'" -->
8     Your connection is properly encrypted. Thank you.
9
10    <!--#elseif expr="$HTTPS = 'off'" -->
11    Your connection is not encrypted.
12    <br>
13    <a href="https://localhost/~adamn/chapter14/felse.shtml" --><!--#echo var="DOCUMENT_URI" -->">
14    Please click here</a> to access this page using an encrypted URL.
15  </p>
16  <!--#endif -->
17 </body>
18 </html>

```



Comparing SSI variables

The following test tests an SSI conditional expression, and displays specific content depending on the test's results.

Your connection is not encrypted.

[Please click here](#) to access this page using an encrypted URL.

Extra

You can nest multiple conditional blocks within each other, just as long as there is the same number of `if` and `endif` elements to correctly open and close them all.

When using strings as test logic, you should enclose the hard-coded text in quotes in a way that does not interfere with the `expr` attribute's parent-quotes. This may involve switching to single-quotes, or by escaping double-quotes with a back-slash (`\`).

Alternatively, you can use forward-slashes instead of quotes around the string text to treat it like a regular expression. SSI actually uses the same regular expression syntax as Perl.

Because the Web server parses SSI, all content within the conditional expression whose test returned false is completely hidden from the end-user. Even if you view the Web site's source code, you cannot access the original source code to the entire conditional expression block. However, if SSI is disabled on the server, the browser renders the SHTML file verbatim, just like a normal HTML file, and all text within the entire conditional expression block is visible!

Display File Statistics with SSI

You can use SSI to access some local file statistics. This feature is limited to displaying only a file's size, or its last-modified timestamp. You can access this feature using the `fsize` and `flastmod` SSI elements. You can identify the file by either the file or virtual attributes. The attribute `file` references the location on the Web server's file-system, while `virtual` references the location on the Web site's domain:

```
<!--#fsize virtual="file" -->
<!--#flastmod virtual="file" -->
<!--#fsize file="file" -->
<!--#flastmod file="file" -->
```

To change the file size units, use the `config sizefmt` argument. It accepts two possible values, "abbrev" and "bytes":

```
<!--#config sizefmt="abbrev" -->
```

Display File Statistics with SSI

1 Open an SHTML file.

2 Type `<!--#fsize virtual="file" -->` to display a file's size in bytes.

Note: Apache may default to either "bytes" or "abbrev" formatting if `config sizefmt` is not yet specified.

3 Type `<!--#flastmod virtual="file" -->` to display a file's last-modified timestamp.

4 Type `<!--#echo var="LAST_MODIFIED" -->` to display the date this page was last modified.

Note: This is a shortcut to using the full `flastmod` command on this SHTML file.

When using "abbrev", the file size is abbreviated by the largest unit value, and then a unit suffix is appended. When using "bytes", just the exact number of bytes on disk displays, with no suffix.

The `flastmod` element uses the default format, "Weekday, dd-Mmm-yyyy hh:mm:ss tz". Use the `config` element and `timefmt` attribute to customize the time format:

```
<!--#config timefmt="%Y-%m-%d %H:%M:%S" -->
```

Note that the system-user account that runs the Apache service requires read-permissions on any referenced files. Alternatively, if you want to display the last-modified date of the user's current Web page, just display the `LAST_MODIFIED` attribute:

This page was last modified on `<!--#echo var="LAST_MODIFIED" -->`

```
1 <html>
2 <body>
3 <h1>Viewing SSI File Stats</h1>
4 The following checks the file size and timestamp of some files on disk.
5 <p>
6 myarchive.zip:<br>
7 <table border=1 cellpadding=5>
8   <tr>
9     <td>
10    <td>
11  </tr>
12 </table>
13 </body>
14 </html>
```

```
1 <html>
2 <body>
3 <h1>Viewing SSI File Stats</h1>
4 The following checks the file size and timestamp of some files on disk.
5 <p>
6 myarchive.zip:<br>
7 <table border=1 cellpadding=5>
8   <tr>
9     <td><!--#fsize virtual="myarchive.zip" --></td>
10    <td><!--#flastmod virtual="myarchive.zip" --></td>
11  </tr>
12 </table>
13 <p>
14 This SHTML file was last modified on:
15 <!--#echo var="LAST_MODIFIED" -->
16 </html>
```

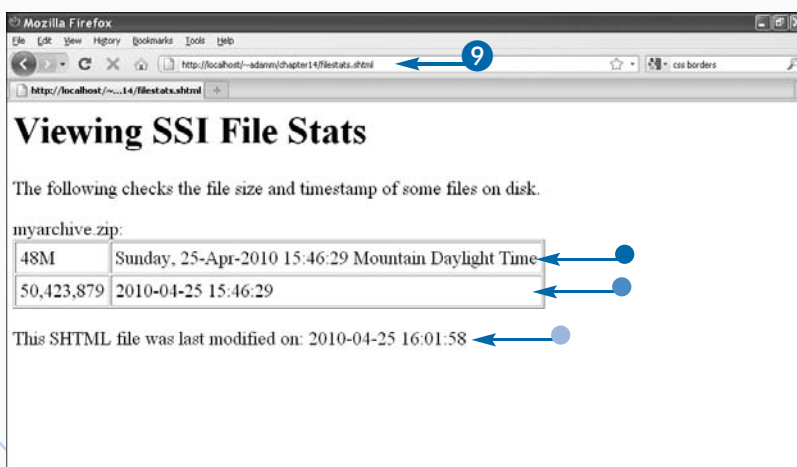
- 5 Type `<!--#config sizefmt="bytes" -->` to change the size that displays to total bytes.
- 6 Type `<!--#config timefmt="%Y-%m-%d %H:%M:%S" -->` to change the time format.
- 7 Copy the `fsize` and `flastmod` tags from steps 2 and 3 to a point after these `config` tags.
- 8 Save the SHTML file.

```

1 <html>
2 <body>
3 <h1>Viewing SSI File Stats</h1>
4 The following checks the file size and timestamp of some files on disk.
5 <p>
6 myarchive.zip:<br>
7 <table border=1 cellpadding=5>
8 <tr>
9 <td><!--#fsize virtual="myarchive.zip" --></td>
10 <td><!--#flastmod virtual="myarchive.zip" --></td>
11 </tr>
12
13
14 <tr>
15 <td>
16 <td>
17 </tr>
18 </table>
19 <p>
20 This SHTML file was last modified on: <!--#echo var="LAST_MODIFIED" -->
21 </body>
22 </html>

```

- 9 In a browser, go to the SHTML file's URL.
 - The file's size and modification date, before using `config`.
 - The file's size and modification date, after using `config`.
 - The timestamp of this SHTML file, using the last configured time format.



Note: All times are relative to the Web server's configured time-zone.

Extra

When customizing the timestamp, you can use the following macros to specify which time-based value and unit you want to display.

MACRO	DESCRIPTION	MACRO	DESCRIPTION
%a	The abbreviated weekday name	%p	"AM" or "PM"
%A	The full weekday name	%P	"am" or "pm"
%b	The abbreviated month name	%r	"a.m." or "p.m."
%B	The full month name	%S	The second (00-59)
%d	The day of the month (01-31)	%T	Same as "%H: %M: %S"
%F	Same as "%Y-%m-%d"	%y	The 2-digit year, without the century
%H	The hour in a 24-hour clock (00-23)	%Y	The 4-digit year, including the century
%I	The hour in a 12-hour clock (01-12)	%z	The time-zone offset from GMT
%m	The month as a number (01-12)	%Z	The time-zone name
%M	The minute (00-59)		

Linking the header, toolbar, and footer templates with a parent SHTML file provides the foundation for a consistent layout throughout the entire Web site. The static HTML defines the unique content to each specific Web page. This is similar to linking the header, toolbar, and footer with dynamic Perl content in that you want to re-use your header, toolbar, and footer throughout the entire site; however, in this case, you have no Perl script providing the main body of content, just static text.

Here you have the freedom to decide whether the template files are generated with Perl and HTML::Template, as described in Chapter 13, or simple SHTML files. Your parent content is still static HTML here, but the template layout can still be dynamically generated.

Ideally, it is best to standardize across the entire Web site how pages are generated. Using SHTML as the *top-level file format* ensures that you can include other SHTML files or CGI files very easily. The top-level file format refers to the actual filename that appears for each unique page in the browser's location field. In other words, you want the user to see filenames such as `index.shtml`, `catalog.shtml`, and `contact.shtml`.

Because the header and footer templates could be either SHTML or CGI, depending on their design requirements, the top-level SHTML file would import them using the `include virtual` or the `exec cgi` SSI element.

You insert the unique content on each top-level page in the middle, in between the header and footer SSI statements. If this content is static, it is the literal HTML code. If it is a dynamic Perl CGI script, you use `exec cgi` to import it.

Link the Header, Toolbar, and Footer with Static HTML Content

- 1 Create a new file for the toolbar.
- 2 Type `<!--#if expr="$page = 'page1' -->class=selected<!--#endif -->` within the *page1* link.
- 3 Repeat step 2 for the other pages, using unique page names.
- 4 Save the toolbar as a SHTML file named `toolbar.shtml`.
- 5 Create a new file for the header.
- 6 Type the header's static HTML code.
- 7 Type `<!--#include virtual="toolbar.shtml" -->` to import a toolbar SHTML file.
- 8 Save the header as a SHTML file named `header.shtml`.

toolbar.shtml

File Edit Tools Syntax Buffers Window Help

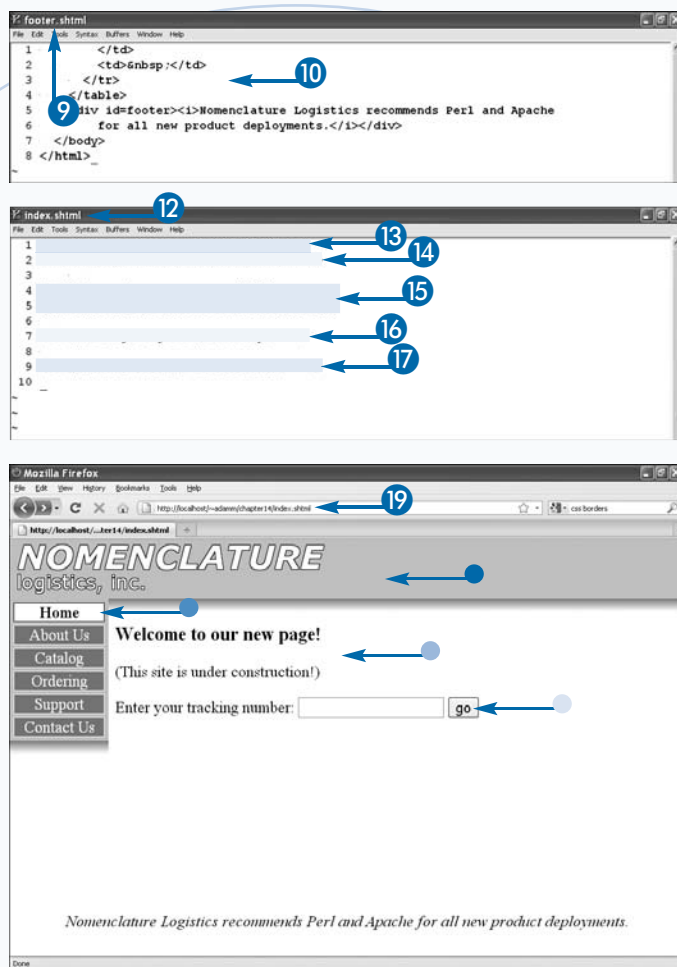
```

1 <table id=toolbar>
2 <tr>
3   <td>
4     <a href="index.shtm">
5     <a href="about.shtm">
6     <a href="catalog
7     <a href="orderi
8     <a href="support
9     <a href="contact
10  </td>
11 </tr>
12 <tr>
13   <td id=shadow>&nbsp;</td>
14 </tr>
15 </table>
16

```

```
1 <html>
2 <head>
3   5 <link rel="stylesheet" href="styles.css" type="text/css">
4   <title></title>
5 </head>
6 <body>
7   <table cellpadding=0 cellspacing=0 border=0 class=main>
8     <tr id=header>
9       <td colspan=2></td>
10      <td align=right>&nbsp;</td>
11    <tr valign=top>
12      <td id=toolbar rowspan=2>
13      <td id=shadow colspan=2></td>
14    </tr>
15    <tr valign=top>
16      <td id=content width=100%>
```


- 9 Create a new file for the footer.
- 10 Type the footer's static HTML code.
- 11 Save the footer as a SHTML file named `footer.shtml`.
- 12 Create a new file for the actual Web page.
- 13 Type `<!--#set var="page" value="pagename" -->` to assign the page name for the toolbar.
- 14 Type `<!--#include virtual="header.shtml" -->` to import the header SHTML file.
- 15 Type some static HTML specific to this page.
- 16 Type `<!--#exec cgi="script.pl" -->` to import a Perl CGI script.
- 17 Type `<!--#include virtual="footer.shtml" -->` to import the footer SHTML file.
- 18 Save the file as a SHTML file named `pagename.shtml`.
- 19 In a browser, open up `pagename.shtml`.
 - The browser displays the header, toolbar, and footer SHTML file content.
 - The correct toolbar link is highlighted.
 - The static HTML.
 - The dynamic Perl script.



Extra

There are three main benefits of following this model of layering SHTML and CGI technologies. First, the final Web site will have a consistent master layout that you can change site-wide by updating a single file. The consistent master layout may react dynamically, depending on what page the user is actually on. For example, the toolbar may highlight *Catalog* when on the `catalog.shtml` page, but the underlying code remains the same site-wide.

Second, you can create new top-level pages very easily, by literally copying the beginning and end of an existing Web page source, and changing the middle. In fact, you may find that a dynamically generated Web page can have only four lines in its top-level file:

```
<!--#set var="page" value="catalog" -->
<!--#include virtual="header.shtml" -->
<!--#exec cgi="cgi/catalog.pl" -->
<!--#include virtual="footer.shtml" -->
```

Finally, you actually obfuscate to the end-user exactly which CGI technology you are using on the site. This can make it more difficult for attackers to target your site; by hiding everything under the guise of an HTML file, and instructing SSI to parse HTML files, you establish the illusion that the entire site is 100-percent static HTML!

Understanding Apache User Authentication

By adding support for user authentication to your Web site, you can secure specific Web pages and directories so they are not available publicly on the Internet. Users must input their personal credentials, which are validated using a chosen authentication model. Depending on the model you use, either Apache or a CGI script will consult with an authorization back-end database to confirm whether those credentials are valid.

Before you can begin, you may want to identify which pages you should restrict, and what level of authorization

a user requires to view them. The easiest way to do this is to follow the browser-based authentication model and create a special subdirectory, such as `http://mydomain.com/private/`. All HTML files, CGI scripts, and images that you place in this directory will be secured by the special access restriction rules as defined in the Apache configuration. Alternatively, you can develop your own Web-based authentication model that is enforced within restricted CGI scripts.

Authentication Workflow

You can authenticate users with Apache and CGI using different authentication models; however, when validating a user, each model follows the same basic workflow. Before you can begin, you need to decide which URLs you want to restrict access to. It may be one or two HTML files, a couple of directories, or the entire domain. Depending on the authentication model you choose, either Apache or CGI needs to be aware of which URLs are restricted.

When a user attempts to access a restricted URL, the Web page should display an *Authentication is required* message and prompt for the user's credentials. If the user's credentials

fail authentication, an appropriate error message should display. Good security design states that you should not be too specific about what failed: for example, if the username is not found or the password is incorrect, you should display a generic message such as *Invalid username or password. Please try again.*

Some more advanced workflow features include a maximum authentication retry timeout, online account registration, and e-mail password recovery. Keep in mind that the browser-based authentication model does not support these features, so you will need to build your own implementation in your CGI scripts.

Browser-Based Authentication

All modern Web browsers support authenticating users with a generic dialog box. First, you must configure Apache to apply a *restricted access control* to a specific directory. That directory leverages an *authentication provider* to validate the supplied username and password credentials. If the credentials are accepted, an *authorization method* is used to check if the user actually has access rights to the directory.

Authentication Types

The authentication type tells the browser how to communicate the user's authentication input to the Web server. Apache supports two main types: basic and digest. Deciding on an authentication type depends on the level of security you want to offer, and the Web browser your users are likely to have.

Basic Authentication Type

Basic authentication means that the user's username and password are sent unencrypted back to the Web server. It is considered the most robust method and is supported by virtually all Web browsers. For local development and testing purposes, it is safe to use basic authentication without an additional security layer. However, it is strongly recommended that you add SSL encryption if you plan on deploying basic authentication onto the public Internet.

Digest Authentication Type

Digest authentication performs just like basic authentication, except that the password is encrypted when it is sent back to the Web server. However, not all Web browsers support digest authentication as an option. Once enabled, an automatic challenge-response communication happens between the browser and the Web server. First, the server sends a random token to the Web browser. The Web browser responds not with the literal password, but with an MD5 hash that combines the username, password, session request, and the original token.

Browser-based Authentication *(continued)*

Authentication Provider

The authentication provider tells Apache how the valid user credentials are stored on the Web server. You must specify exactly one authentication provider per secured URL directory in Apache. Sourcing from multiple authentication providers within the same restricted URL directory is not possible.

Static Password File

The Web site administrator creates and maintains a static password file, which you can use as an authentication provider. It contains all of the authorized users and passwords to a particular secure area of the site. This file cannot be accessed directly by URL, but instead by Apache using the active authentication type. The administrator uses the program `htpasswd` or `htdigest`, depending on the authentication type; they can use the program to add, update, and remove credentials from the file.

Static Database File

Using a static database file as your authentication provider is very similar to the static password file method, except that the database is much more efficient at managing a large number of credentials in a single file. The administrator uses the program `htdbm` to add, update, and remove DBM-formatted credentials from the database file.

SQL Database

You can use a custom SQL database as an authentication provider. This is especially useful if you already have a list of usernames and passwords in a database, and you want to instruct the Apache basic or digest authentication types to support it. You can configure a custom SQL statement specific to your user's table that Apache will execute to access the password.

Local Users

You can use the local user accounts on the Web server as an authentication provider with the help of the pluggable authentication modules framework called PAM. This method is only available on Unix systems where users already have a need for a local account, such as an SSH or FTP access requirement, and you want to allow them to use their local user credentials to authenticate on to your Web site.

Authorization Method

The authorization method determines if a user has specific rights to a secure area, regardless of whether their credentials have passed authentication. This allows you to set up multiple secure directories that allow different combinations of user access.

Allow Specific Users

You can define specific users in the Apache `config` file or `.htaccess` file as being allowed access to a secure URL. Apache checks for authorization after their credentials have passed the authentication stage.

Allow Specific User-Groups

You can bundle individual users together into logical groups and authorize them based upon group membership. The Apache `config` file or `.htaccess` file must reference the authorized group name. Apache checks for group authorization after their credentials have passed the authentication stage.

Secure a Directory Path with Apache

You can configure Apache to secure a directory from public access. Doing so will cause the user's Web browser to display a generic dialog box, prompting them for a username and password. You have a choice of how to instruct Apache about which directory is to be restricted. You do this by applying special authentication directives within either a *directory context* or an *.htaccess context*.

The directory context implies editing the Apache `httpd.conf` configuration file and creating a `<directory path>` configuration group. All files and subdirectories within the specified directory are treated as restricted and will require authorization.

The `.htaccess` context means you can create a special file called `.htaccess` directly within the directory you want to restrict. You can write the authentication

directives to this file. All files and subdirectories stored under the `.htaccess` directory are treated as restricted and will require authorization. You need to restart Apache to apply directory context changes, but not `.htaccess` changes. To activate the `.htaccess` file, you must enable the `AllowOverride` directive's `AuthConfig` option. You need to apply this onto a specific directory context, which is why you must restart Apache.

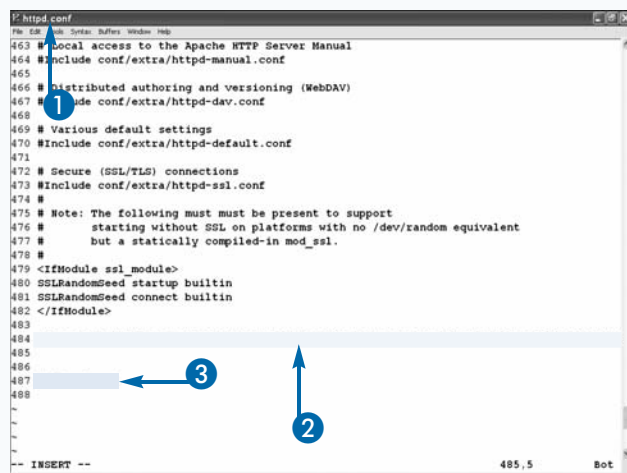
You use the authentication directives to specify which authentication type, authentication provider, and authorization method you want to apply. The easiest authentication type and provider to use is the Basic file:

```
AuthType Basic
AuthName name
AuthBasicProvider file
```

After you do this, the next step is to create the password file that contains the users' credentials.

Secure a Directory Path with Apache

- 1 Open the Apache configuration file.
- 2 Type `<Directory path>` to include the full directory that you want to restrict access to.
- 3 Type `</Directory>` to close off the directive.

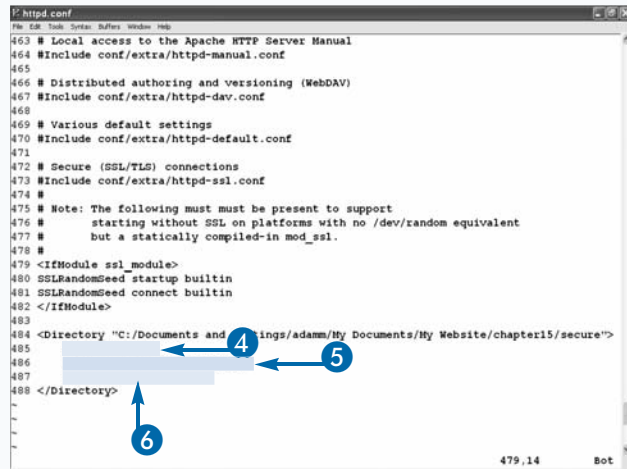


The screenshot shows the `httpd.conf` file in a text editor. The file contains various configuration directives. A blue box highlights the area where the `<Directory>` directive is being added. A blue arrow points to the `<Directory>` directive, and another blue arrow points to the `</Directory>` directive. The status bar at the bottom shows line 485, 5.

- 4 Type `AuthType Basic`.
- 5 Type `AuthName name`.

Note: The `AuthName` value is merely a label to the secure area. It will be displayed to the user in the login popup message.

- 6 Type `AuthBasicProvider file`.



The screenshot shows the `httpd.conf` file in a text editor. The file contains various configuration directives. A blue box highlights the area where the `AuthType`, `AuthName`, and `AuthBasicProvider` directives are being added. A blue arrow points to the `AuthType` directive, and another blue arrow points to the `AuthName` directive. The status bar at the bottom shows line 479, 14.

7 Type **<Directory path>**, inserting a different restricted directory path.

8 Type **AllowOverride AuthConfig**.

Note: Configuring AllowOverride allows you to secure a directory with a special .htaccess file. This is an alternative to editing the master configuration file and restarting the Apache server to apply the change.

9 Type **</Directory>** to close off the directive group.

10 Save the Apache configuration file and restart Apache.

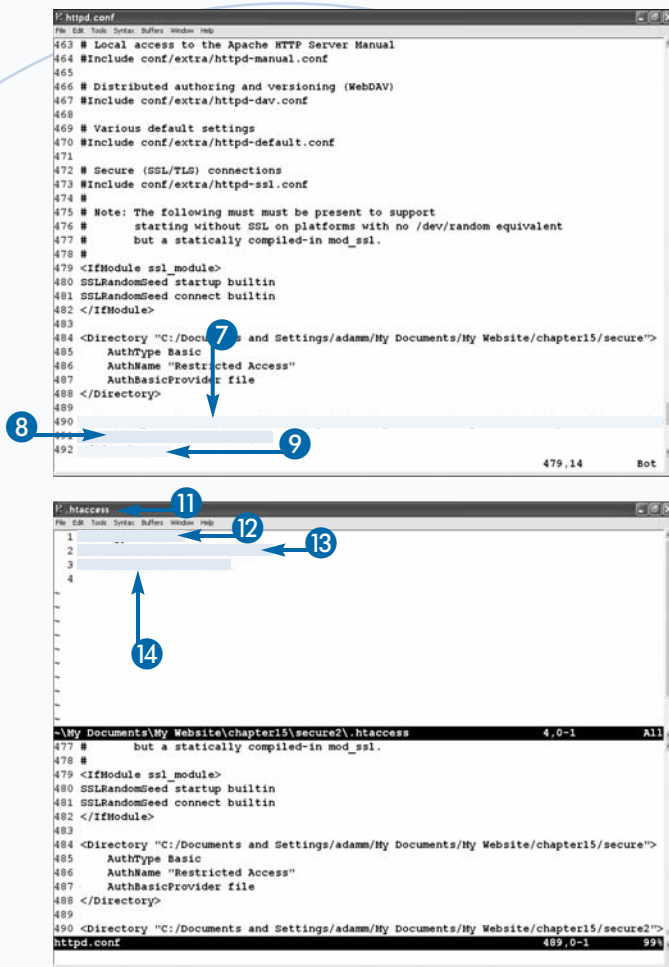
11 Create a new file called **.htaccess** in the directory path you specified in step 7.

12 Type **AuthType Basic**.

13 Type **AuthName name**.

14 Type **AuthBasicProvider file**.

15 Save the **.htaccess** file.



Extra

Digest authentication allows for a new field for authentication, the *realm*, which the end-user must type along with their username and password. Allowed usernames, passwords, and realms are specified in the digest authentication password file:

```
AuthType Digest
AuthName name
AuthDigestProvider file
AuthDigestDomain URL
```

The directory context in the Apache configuration file allows for some alternative ways to specify a directory path that you want to secure. Instead of using `<directory path>`, you can use `<location path>` to specify a directory from root of the domain, and `<files filename>` to restrict by a specific filename or wildcard expression:

```
<location /private>
  AuthType type
  AuthName name
  ...
</location>
```

For example, to restrict a specific file-naming convention, you can use `<files "-private.html">...</files>`. Unlike `<directory>` and `<location>`, you can also use `<files>` within the `.htaccess` context. This allows you to specify matching files within the specific directory to which the `.htaccess` file is written.

Use an Authentication Password File

You can configure Apache to handle basic or digest authentication using a password file as your authentication provider. This file stores all end-user credentials that can be used for authentication. If you are using basic authentication, you need to use the `htpasswd` command. This accepts the password file and the username as arguments, and then prompts you for a password:

```
htpasswd passwordfile username
```

If you are using digest authentication, you must use the `htdigest` command. This is just like the first command, except it requires a `realm` argument:

```
htdigest passwordfile realm username
```

If the `passwordfile` does not yet exist, you must use a `-c` argument to create it. However, do not use `-c` on an existing password file, as the entire file will be truncated!

Use an Authentication Password File

- 1 Open a Terminal window.
- 2 Go to a directory to hold the password file outside any visible Web site directory.

Note: In Unix, the path `/etc/apache2/passwords` is acceptable.

- 3 Type `htpasswd -c passwordfile username` and press Enter. (In Windows, use the full path to the `htpasswd.exe` file.)

Note: If you used digest authentication, use `htdigest -c passwordfile realm username` instead.

Note: If the program is not in your path, you need to specify the location of the binary.

- 4 Type in the new user's password.
 - 5 Type the password in again to confirm it.
 - 6 Display the password file's contents.
- The user's credentials have been written to the password file.

Because Apache has either a Basic file or a Digest file configured as the authentication type and provider, you need to add a new configuration directive called `AuthUserFile` to tell Apache where to find the password file on the Web server:

```
AuthUserFile passwordfile
```

These password management programs are part of a standard Apache installation on most Windows and Unix installations, but they may not be in your system path. On Windows, you should find them under the default directory, `C:\Program Files\Apache Software Foundation\Apache2.2\bin`. If you use Debian or Ubuntu Linux, you may need to install the "apache2-utils" package, which provides these programs. After you establish the password file, the final step is to configure the authorization method.

```
Command Prompt
C:\Documents and Settings\adam\My Documents\My Website\chapter15>
```

```
Command Prompt
C:\Documents and Settings\adam\My Documents\My Website\chapter15>cd ../../
C:\Documents and Settings\adam\My Documents>htpasswd -c passwordfile username
Automatically using MD5 format.
New password:
Re-type new password:
Adding password for user testuser
C:\Documents and Settings\adam\My Documents>
testuser:$apr1$NrpVX7hN$Ak2fBYjt5yWk0k465yQ9u/
C:\Documents and Settings\adam\My Documents>
```


Require Only Authorized Users

After you configure Apache for restricted access control by securing a directory path, and you have enabled an authentication provider like a password file, the last step is to apply an authorization method. If your Web site has only one password-protected directory and one password file, you may simply want to configure Apache to treat all users who successfully authenticate as authorized users. To do this, you add the `Require valid-user` directive into the Apache directory or `.htaccess` context configuration. However, if your Web site has several password-protected directories using a single users' passwords file, and a mix of which users are allowed into each directory, then you need to use `Require` to assign authorized users: `Require user userid1 userid2 userid3 [...]`.

It may not be practical to explicitly include each user for every secured directory they have access to. To simplify things, you can assign each user into a logical group, and then assign one or more groups as having authorized access:

```
AuthGroupFile groupfile
Require group groupname1 groupname2 groupname3
[...]
```

The group file follows a very simple format, and you can create it manually:

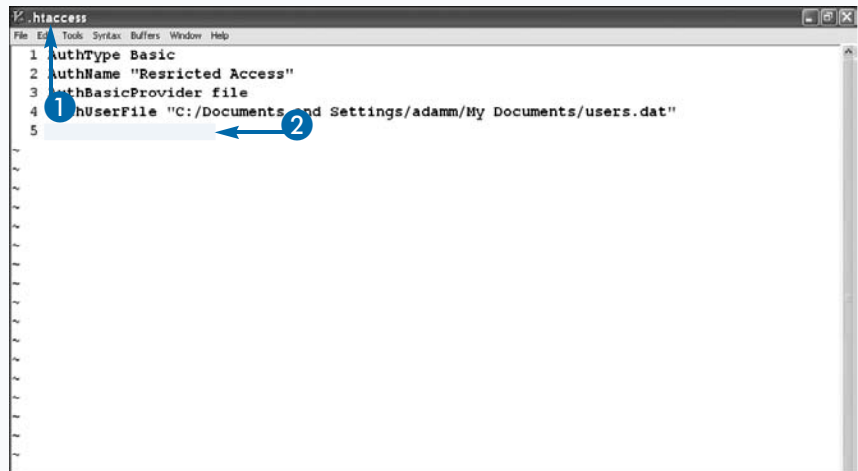
```
groupid1: userid1 userid2 userid3 [...]
groupid2: userid1 userid2 userid3 [...]
[...]
```

All three uses of the `Require` directive can be mixed and matched in a single directory; however, the `valid-user` option makes the other two moot. At a minimum, you need to configure at least one `Require` directive to complete the authentication and authorization configuration process.

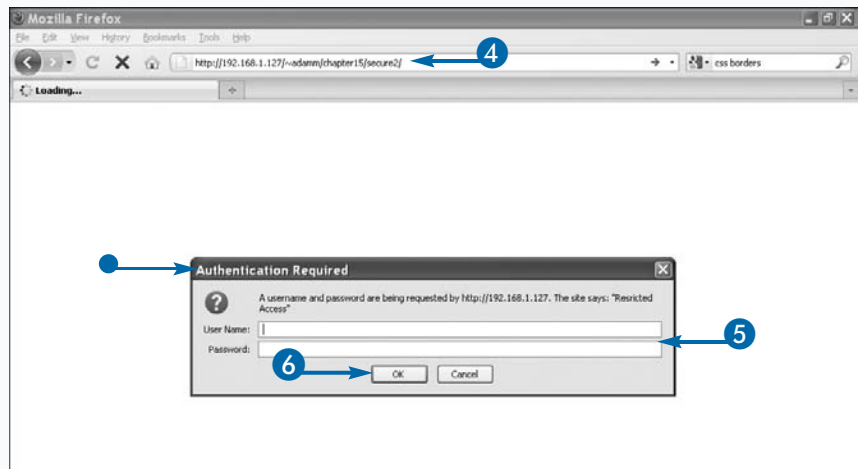
Require only Authorized Users

- 1 Open the Apache configuration file, or the secure directory's `.htaccess` file, in a text editor.
- 2 Type **Require valid-user**.
- 3 Save the file.

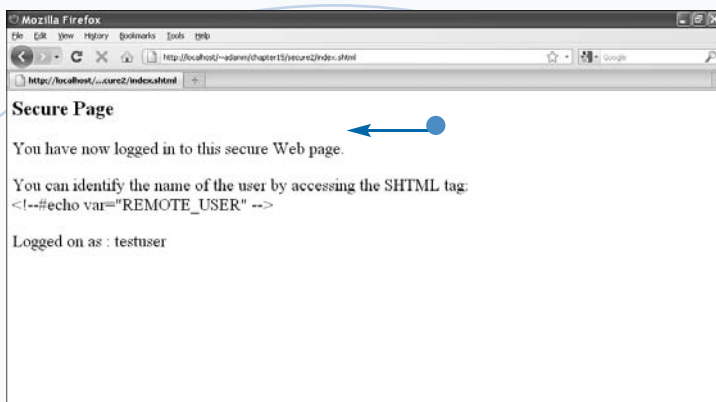
Note: If you modified the Apache configuration file, you need to restart the Web server.




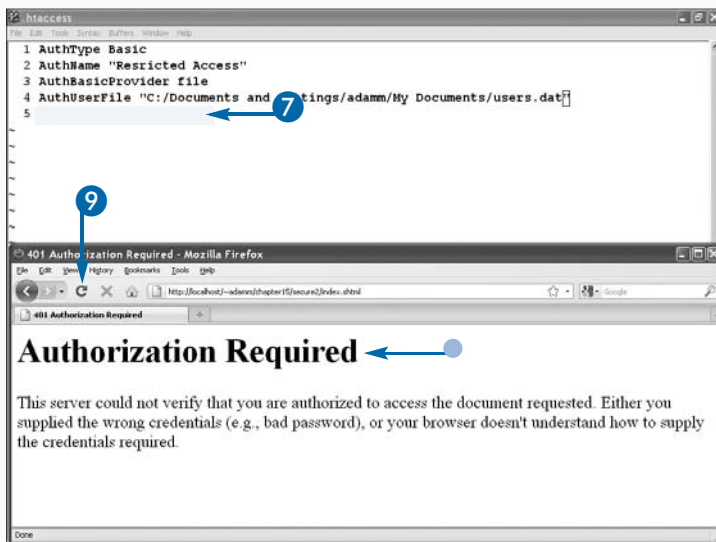
- 4 Open a Web browser to the secure URL.
 - The login prompt appears.
- 5 Type in a user's credentials that you saved in the password file.
- 6 Click OK (or Log In, depending on your Web browser).



- The correct Web page appears in the browser.



- 7 Type **Require user username**.
- 8 Save the file and restart Apache if necessary.
- 9 Click  to reload the Web page.
 - An authorization error appears. Although the user has authenticated, they are no longer authorized.



Apply It

If you have a fairly large user or group list, you can construct either file using the DBM format. This will make searching more efficient for Apache when enforcing credentials. You must use the `dbmmanage` program to manage DBMs. This program is more complex than `htpasswd` and `htdigest`, and its syntax is as follows:

```
dbmmanage dbmfile command username password groupname1,groupname2,...
```

Basic *command* options are `add`, `delete`, or `update`. The *password* argument accepts either a period (.) to prompt for a new password, or a dash (-) to keep the previous password. The *groupname* arguments should be in a comma-separated list. For example, you can create the user `jsmith`, prompt for his password, and assign him into the group's `users` and `developers` with the following code:

```
dbmmanage dbmfile add jsmith . users,developers
```

You can remove `developers`, add `managers`, and keep the previous password with the following code:

```
dbmmanage dbmfile update jsmith - users,managers
```

You must instruct Apache where to find your DBM file:

```
AuthDBMUserFile dbmfile
AuthDBMGroupFile dbmfile
```

Understanding User Authentication in Perl



If you are not satisfied with the Web browser's generic login prompt, you have the option to develop your own authentication logic and HTML prompt design. You do this by creating a new Perl Authentication module, which in turn is leveraged within any CGI script that requires restricted access.

You will control all identification, authorization, and validation functionality. This method is recommended for

larger, more complex Web sites as the user ID and password prompts are free-form HTML. You can embed them in the site's layout, creating a more professional look than a generic Web browser popup prompt.

Developing your own Perl Authentication module means that only Perl CGI code can be secured. You cannot directly limit access to a group of static HTML pages, or an entire directory or domain.

Authentication Workflow

Authenticating users with a Perl CGI module requires managing the authentication process. The same module will be loaded multiple times; you need to be aware of the user's current state every time a restricted Perl CGI page is loaded.

This module is responsible for tracking the user, displaying a login prompt, processing the user's credentials, and authorizing the user. When finished, it returns control to the

parent CGI script, allowing the script to execute if authorized, or exiting prematurely if not.

You can develop some more advanced workflow features, such as a maximum authentication retry timeout, on-line account registration, and e-mail password recovery, at any time. They act as supplementary features that complement the Authentication module.

Perl-Based Authentication

You can use Perl to secure any Perl-based dynamic CGI page on your Web site through a custom Authentication module. This module would be in charge of displaying a login prompt, accepting the user's input, validating the user's credentials, identifying when the user's session is active, and allowing a secured CGI script to display normally.

Secure Perl CGI Scripts

When securing an individual Perl CGI script, you want the script to execute, but without the normal script functionality. One way to implement this is to import a special Authentication module into the Perl CGI script.

This module provides the logic that validates the user's session, and displays a login prompt by way of a special startup method. The CGI calls this method early in its execution, and watches for the return value. If it returns a failure code, it means that the user has not been authenticated to view the page; the CGI script detects the failure and exits its normal process flow early.

If the validation method returns a success code, the CGI script recognizes that the module has authenticated the user, so it is allowed to continue on and execute its designed functionality.

Perl Authentication Module

When used within any Perl CGI script, the Perl Authentication module restricts access to that script. The module is responsible for the following authentication requirements: tracking the user's session identifier, authorizing their session, prompting for credentials, and validating the credentials against a database.

Every time that CGI script is loaded, the all four authentication requirements are processed. If the session still has authorized access, there is no need to prompt the user again for their credentials; the Authentication module is completely transparent, allowing active sessions to execute the CGI script.

Perl-based Authentication *(continued)***Session ID Cookie**

A session identifier is a unique token that is stored as a cookie by the user's Web browser. If the session is undefined, the module assigns a new cookie, which may expire when the browser is closed, effectively logging out the user's session, or after a set timeout. The session identifier format should not be a simple incrementing number, but should have some element of randomness to it. It should also be a value that no user would be able to guess simply by changing one or two numbers.

Authorizing an Existing Session

If the Authentication module receives a session cookie back from a visiting browser, it means that the user has been here before. The user may have an already existing session, in which case you do not need to re-authenticate their credentials. The module searches for the session identifier in its user database. If it finds one that matches, it knows who this user is and allows the normal CGI script to execute.

Optionally, the user database may implement some sort of timestamp indicating the last time the authorized user visited the page. Your Web site may implement a form of inactivity-timeout by checking against this timestamp and validating that the user has not been away from the site for too long.

Naturally, if the current time is close enough to the session's timestamp, the user is still valid. You need to update the timestamp to the current time and reset their time-out clock. If authorized, the module exits with a success return code, and the normal CGI script is executed.

HTML Login Prompt

Whenever authorization fails, an HTML form appears that prompts the user for a username and password. The Authentication module must present this HTML form instead of the regular CGI script's output, because access to the script is restricted and you need to validate the user's credentials.

The actual CGI also must accept the HTML form's input. This is usually done by submitting the username and password in the form. The Authentication module, recognizing that the session is still invalid, identifies that credentials were passed as form input. It takes that input and compares the username and password against a user database. Because the module recognizes that the session is not authorized, it exits with a failure code. This inhibits the parent CGI script from continuing its normal execution.

Validating a User's Credentials

The validation process needs to compare the supplied username and password against the database. If it finds a match, it authorizes the session. If not, it displays the login prompt again, with an appropriate error message.

The password should be encrypted or hashed in some form during this validation process. There is never a good reason to store the user's password in clear text in the database; it is bad enough to receive the password in clear text over HTTP. At least SSL mitigates this problem.

Authorizing a New Session

If the module identifies the supplied credentials match the database, then the current session identifier is written into the user's record. This now acts as the lookup value that the module uses for authorizing an existing session. The module exits with a success return code, and the parent CGI script continues with its normal functionality.

Create a Perl Authentication Module

You can create your own Perl Authentication module to have direct control over the authentication, authorization, and validation of users on your Web site. This is useful because you can create your own custom HTML form that appears within the Web site. This produces a more professional-looking Web site than a generic browser popup window. Once it is completed, you can add the module to any CGI script, effectively restricting access to the script's logic, and forcing the user to authenticate. When a user accesses a restricted CGI script, the module initializes, and its validation logic begins.

First, the module must track each user by a session identifier cookie. A new session cookie is assigned for new users. Returning users keep the same cookie, until the cookie expires when they close the Web browser.

Create a Perl Authentication Module

- 1 Create a new Perl module called `Auth.pm`.
- 2 Import the CGI module and initialize a handle to it when the Authentication module is loaded.
- 3 Load the session cookie into a session identifier variable when the module is loaded.
- 4 If absent, assign the variable a new session value, and print the cookie HTTP header.

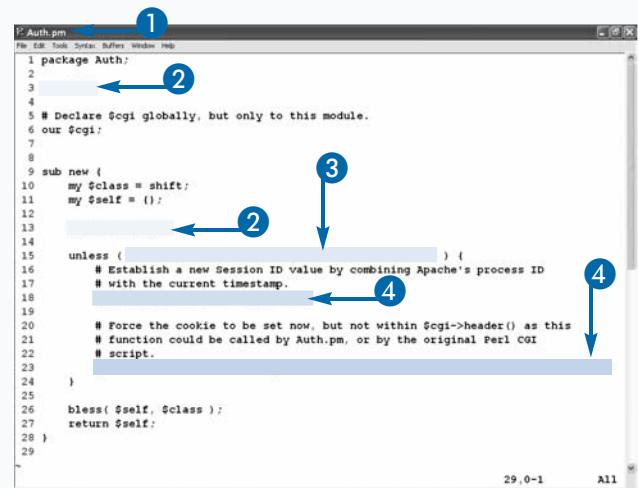
- 5 Create a new subroutine called `validate`.
- 6 Call a subroutine that checks the session identifier for pre-existing authorization in the database.
- 7 If authorized, type **return 0**; to stop the `validate` subroutine with a success code.
- 8 Call a subroutine that displays an HTML login prompt.
- 9 Type **return 1**; to stop the `validate` subroutine with a failure code.

Note: The parent CGI script will act upon these return values later.

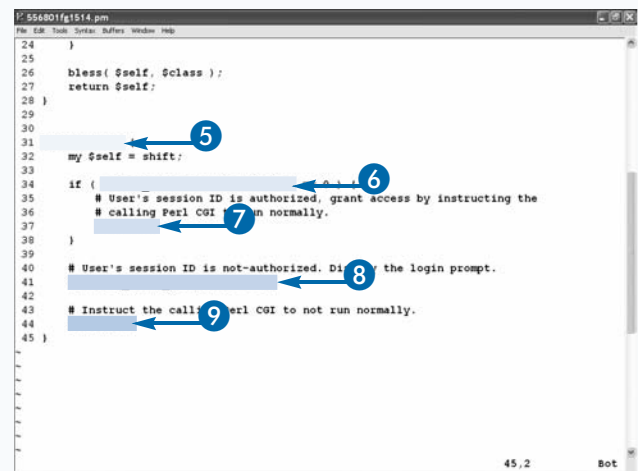
Second, the session cookie is checked against a persistent database. If a user record is found that matches the identifier, the session is authorized and the module exits. The original CGI script is allowed to continue normally.

If no user record is found, the user is not currently authorized and must be authenticated. This process happens in four steps within the module's `validate` subroutine: the module checks for session authorization, displays a login prompt, validates the user's credentials, and finally authorizes the user's session.

Each time the user loads the restricted CGI script, the same four-step procedure is followed. This ensures that the user always has a valid session while accessing a secure area of your Web site. These four steps are demonstrated in this chapter.



```
1 package Auth;
2
3
4
5 # Declare $cgi globally, but only to this module.
6 our $cgi;
7
8
9 sub new {
10     my $class = shift;
11     my $self = {};
12
13     unless ( ) {
14         # Establish a new Session ID value by combining Apache's process ID
15         # with the current timestamp.
16
17         # Force the cookie to be set now, but not within $cgi->header() as this
18         # function could be called by Auth.pm, or by the original Perl CGI
19         # script.
20
21         bless( $self, $class );
22         return $self;
23     }
24
25     bless( $self, $class );
26     return $self;
27 }
28
29
30
```



```
24
25
26 bless( $self, $class );
27 return $self;
28
29
30 my $self = shift;
31
32 if (
33     # User's session ID is authorized, grant access by instructing the
34     # calling Perl CGI to run normally.
35 ) {
36
37     # User's session ID is not-authorized. Display the login prompt.
38
39     # Instruct the calling Perl CGI to not run normally.
40
41
42
43
44
45 }
```


- 10 If not authorized, check for any submitted HTML form data.
- 11 If there is data, call a function to validate the submitted credentials against the database.
- 12 If validated, call a subroutine that authorizes the user's session identifier.
- 13 Type **return 0**; to stop the subroutine with a success code.
- 14 If no HTML form data is found, or if validation fails, continue on to the HTML login prompt and failure return code.

Scroll to the bottom of the `Auth.pm` file.

- 15 For each of the four calls made during `validate`, create new placeholder subroutines.

Note: These subroutines will be populated in sections “Check for Session Authorization”, “Display a Login Prompt”, “Validate a User’s Credentials”, and “Authorize a User’s Session” in this chapter.

- 16 Type **1**; on the very last line of the file.
- 17 Save the Perl module.

```

24 }
25
26 bless( $self, $class );
27 return $self;
28 }
29
30 sub validate {
31     my $self = shift;
32
33     if ( $check_authorization( $self ) == 0 ) {
34         # User's session ID is authorized. grant access by instructing the
35         # calling Perl CGI to run normally.
36         return 0;
37     }
38
39     # The user is trying to login. Process the credentials.
40     if (
41         # Credentials check out. Add authorization to grant access.
42         # Credentials failed. Continue with displaying the login prompt again.
43     ) {
44         # User's session ID is not-authorized. Display the login prompt.
45     }
46
47     # Instruct the calling Perl CGI to not run normally.
48     return 1;
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81

```

```

52 return 1;
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81

```

Extra

While you are working on your Perl module, you can validate the code that is being produced by running it as a normal Perl script on the command-line:

```
perl Auth.pm
```

There is no output, but at least you can quickly validate if there are any syntax errors. After you link the module with an actual Perl CGI script, any errors in the module actually appear as a blank Web page in your browser. The Apache error logs will contain any error messages, but to make accessing that information available in the Web browser, you can add the `CGI::Carp` module directly into this module:

```
use CGI::Carp qw( fatalToBrowser );
```

For more information on `CGI::Carp`, see Chapter 12.

Access a User's Database

The Authentication module requires access to a persistent database on the Web server.

The users' database stores usernames, password hashes, active session identifiers, and timestamps. The Authentication module will use this information later to process who is currently authorized, who can be authenticated, and whose session has expired due to inactivity.

In this example, you use the Storable module's `lock_store` and `lock_retrieve` methods. The Storable module allows you to easily convert complex Perl variables into binary data on disk, and retrieve that data again. In this case, you will be creating a special hash reference within the actual module to keep track of your database while loaded in memory.

Access a User's Database

- 1 Type `use Storable qw(lock_store lock_retrieve);` to import the Storable module's subroutines as functions.
- 2 Type `use constant { USERDB_FILE => "filename" };` to link the constant `USERDB_FILE` to the full path to the user database file.
- 3 Type `-e USERDB_FILE` to test if the file actually exists on disk.
- 4 Type `$self->{'db'} = lock_retrieve(USERDB_FILE);` to load the user database into memory.

Note: If you try to load a file that does not exist, `lock_retrieve` will error out and kill the module.

You create the database hash reference in your new subroutine. Its first job is to use `lock_retrieve` and load the file-system's copy of the file into memory, or to create it with `lock_store` if it does not yet exist. Within the module, the database is accessible through the `$self` hash ref. `$self` refers to the current state of the module's instance, from the module's perspective:

```
$self->{'db'} = database;
```

User passwords are hashed, and will be stored under a "users"→username key:

```
$self->{'db'}->{'users'}->{username} = password_hash;
```

Session-specific information is stored under a "sessions"→session-ID key:

```
$self->{'db'}->{'sessions'}->{session_ID} = {  
    user => user_reference,  
    lastseen => timestamp };
```

```
1 package Auth;  
2  
3 use CGI;  
4  
5  
6  
7 # Declare $cgi globally, but only to this module.  
8 our $cgi;  
9  
10  
11 sub new {  
12     my $class = shift;  
13     my $self = {};  
14  
15     $cgi = new CGI;  
16  
17     unless ( $self->{'sessID'} = $cgi->cookie( 'sessID' ) ) {  
18         # Establish a new Session ID value by combining Apache's process ID  
19         # with the current timestamp.  
20         $self->{'sessID'} = $$ . time;  
21  
22         # Force the cookie to be set now, but not within $cgi->header() as this  
23         # function could be called by Auth.pm, or by the original Perl CGI  
24         # script.  
25         print "Set-cookie: " . $cgi->cookie( 'sessID', $self->{'sessID'} ) . "\n";  
26     }  
27  
28     bless( $self, $class );  
29     return $self;  
30 }
```

```
1 package Auth;  
2  
3 use CGI;  
4 use Storable qw( lock_store lock_retrieve );  
5 use constant { USERDB_FILE => "../../userdb.dat" };  
6  
7 # Declare $cgi globally, but only to this module.  
8 our $cgi;  
9  
10  
11 sub new {  
12     my $class = shift;  
13     my $self = {};  
14  
15     $cgi = new CGI;  
16  
17     if (   
18  
19 )  
20  
21     unless ( $self->{'sessID'} = $cgi->cookie( 'sessID' ) ) {  
22         # Establish a new Session ID value by combining Apache's process ID  
23         # with the current timestamp.  
24         $self->{'sessID'} = $$ . time;  
25  
26         # Force the cookie to be set now, but not within $cgi->header() as this  
27         # function could be called by Auth.pm, or by the original Perl CGI  
28         # script.  
29         print "Set-cookie: " . $cgi->cookie( 'sessID', $self->{'sessID'} ) . "\n";  
30     }
```

- 5 If the file does not yet exist, initialize the users and sessions keys to blank hash refs.
- 6 Type `lock_store($self->{'db'}, USERDB_FILE);` to update the database file on disk.

```

1 package Auth;
2
3 use CGI;
4 use Storable qw( lock_store lock_retrieve );
5 use constant { USERDB_FILE => "../userdb.dat" };
6
7 # Declare $cgi globally, but only to this module.
8 our $cgi;
9
10
11 sub new {
12     my $class = shift;
13     my $self = {};
14
15     $cgi = new CGI;
16
17     if ( !-e USERDB_FILE ) {
18         $self->{'db'} = lock_retrieve( USERDB_FILE );
19     }
20     else {
21         # [Redacted]
22     }
23 }
24
25 # [Redacted]
26
27 }
28
29 unless ( $self->{'sessID'} = $cgi->cookie( 'sessID' ) ) {
30     # Establish a new Session ID value by combining Apache's process ID
31     # with the current timestamp.
32 }
  
```

- 7 Scroll to the bottom of the file.
 - All support functions accept `$self`, so they will all have access to `$self->{'db'}`.
- 8 Save the Perl module.

```

65 return 1;
66
67 }
68
69 sub check_authorization {
70     # Force invalid authorization until logic can be written.
71     return 1;
72 }
73
74
75 sub check_credentials {
76     # Force invalid credentials until logic can be written.
77     return 1;
78 }
79
80
81
82 sub add_authorization {
83     # Nothing to do
84 }
85
86
87
88 sub display_login_prompt {
89     # Nothing to do
90 }
91
92
93
94 1;
  
```

Extra

The Storable module is not the only way to create a database. However, it is one of the simpler ways, as it easily converts between any type of Perl variable and binary data. This means that you only need to construct a so-called *database* with nested hash and array references to have an instantly indexing table.

This method is good enough for demonstrating an overlying concept, such as this Authentication module, but if you implement an Authentication module on an actual Web site, then a proper relational database is strongly recommended.

One popular database program, MySQL, is discussed in Chapter 21. If you were to apply its Perl API into your Authentication module, the majority of the logic code discussed in this chapter would actually stay the same.

Implementing is rather easy. First, when the module is initializing, you need to call the *connect* API to create a link to the database back-end. Second, each of the support functions that need database access, such as *check_authorization*, *check_credentials*, and *add_authorization*, will have to query and update the database with a special language called SQL.

You will find that a relational database results in better overall performance and CPU efficiency, especially when dealing with large amounts of data and traffic.

Store User Credentials in a User's Database

You need to have the means to write user credentials into a database so that the Authentication module can both identify who is allowed to log in, and have access to your secure password data. This user-database program could be a command-line Perl script that is executed outside of CGI. You do not want anyone to type in a URL to access this program. Here you are leveraging the Storable module again, using the same `lock_retrieve` and `lock_store` functions that the Authentication module uses. This maintains the exclusivity lock on the file while it is in use. The `SHA1::Digest` module will be used as a hashing algorithm in order to store valid passwords. Later in this chapter, the section “Validate a User's Credentials” will also use the same hashing algorithm to compare the user's submitted password-hash to the database's stored password-hash.

Store User Credentials in a User's Database

- 1 Create a new Perl script.
- 2 Type `use constant { USERDB_FILE => "filename" };` to link the constant `USERDB_FILE` to the full path to the actual user database file.
- 3 Type `use Storable qw(lock_store lock_retrieve);` to import the Storable module's subroutines as functions.
- 4 Type `$db = lock_retrieve(userfile);` to load the user's database into memory.
- 5 Read the first command-line argument as `$cmd`; otherwise, display the help screen.
- 6 Type `use Digest::SHA qw(sha1_base64);` to import a data-hashing module.
- 7 Check if `$cmd` is "add"; otherwise, display the help screen.
- 8 Read the next two arguments as the username and password.
- 9 Type `$db->{ 'users' }->{ lc $username }->{ 'pwhash' } = sha1_base64($password);` to hash passwords and store them in `$db`.
- 10 Type `lock_store($db, userfile);` to write the database to disk.

When running the program, the first argument is the command, followed by a series of options. If an unknown command is provided, or the wrong number of options, a help screen should display as output:

```
userdb.pl command options ...
```

The first command to implement is `add`. It accepts two arguments: a username and a password. The second command to implement is `dump` with no arguments. It allows you to see the database contents:

```
userdb.pl add username password
userdb.pl dump
```

Naturally, accepting a password in clear text is not the best idea. Fortunately, it is possible to program Perl to prompt you for the password and not echo the characters back to the screen. You never know who could be looking over your shoulder.

```
1 #!C:\Perl\bin\perl.exe
2
3
4
5
6
7 my $db = {};
8 if ( -e USERDB_FILE ) {
9     $db = lock_retrieve( USERDB_FILE );
10 }
11
12
13
14 sub ShowHelp {
15     print "Usage:\n";
16     print "\t$0 <command> [options]\n";
17     print "\n";
18     print "Commands:\n";
19     print "\tadd\tadd new users to the database\n";
20     print "\toptions: username password\n";
21     print "\n";
22     print "\tdump\tDump the database contents\n";
23     print "\n";
24     exit 0;
25 }
```

```
26
27
28 use constant { USERDB_FILE => "userdb.dat" };
29 use Storable qw( lock_store lock_retrieve );
30
31
32 my $db = {};
33 if ( -e USERDB_FILE ) {
34     $db = lock_retrieve( USERDB_FILE );
35 }
36 my $cmd = shift( @ARGV ) || ShowHelp();
37
38
39
40
41
42 else {
43     ShowHelp();
44 }
45
46 sub ShowHelp {
47     print "Usage:\n";
48     print "\t$0 <command> [options]\n";
49     print "\n";
50     print "Commands:\n";
51     print "\tadd\tadd new users to the database\n";
52 }
```

- 11 Type **use Data::Dumper;** to import the module.
- 12 Test if `$cmd` is "dump".
- 13 Type **print Dumper(\$db);** to display the contents of `$db`.

Save the Perl script as `userdb.pl`.

```

1 #!C:\Perl\bin\perl.exe
2
3 use constant { USERDB_FILE => "userdb.dat" };
4 use Storable qw( lock_store lock_retrieve );
5 use Digest::SHA qw( sha1_base64 );
6
7
8 my $db = {};
9 if ( !e USERDB_FILE ) {
10     $db = lock_retrieve( USERDB_FILE );
11 }
12 my $cmd = shift( @ARGV ) || $ShowHelp();
13
14 if ( $cmd eq "add" ) {
15     my $username = shift( @ARGV ) || $ShowHelp();
16     my $password = shift( @ARGV ) || $ShowHelp();
17     $db->{ 'users' }->{ lc $username }->{ 'pwhash' } = sha1_base64( $password );
18     lock_store( $db, USERDB_FILE );
19 }
20
21
22
23
24 }
25 else {
26     $ShowHelp();
27 }
28
29 sub ShowHelp {
30     print "Usage:\n";

```

- 14 Open a Terminal window.
- 15 Type **userdb.pl** and press Enter.
 - The help screen appears.

- 16 Type **userdb.pl add username password** and press Enter.

The username and password are written to the database.

- 17 Type **userdb.pl dump** and press Enter.

- The Terminal window displays the database contents: all stored users and their hashed passwords.

```

C:\Documents and Settings\adama\My Documents> userdb.pl
Usage:
  C:\Documents and Settings\adama\My Documents\userdb.pl <command> [options]

Commands:
  add      Add new users to the database
           Options: username password
  dump     Dump the database contents

C:\Documents and Settings\adama\My Documents> userdb.pl add testuser test
C:\Documents and Settings\adama\My Documents> userdb.pl dump
$VAR1 = {
    'users' => {
        'testuser' => {
            'pwhash' => '1E3a8a3TUZSPys4VWu290nSskjk'
        }
    }
};
C:\Documents and Settings\adama\My Documents>

```

Apply It

While this program may seem rather crude, it serves its purpose. Basically, you are providing a means to manipulate a database file and store passwords securely. Eventually, you could create an administrative CGI script that would allow you to manipulate the list of authorized users, directly from your Web browser. Before you get too far, it is a good idea to add some additional functionality to this program, such as updating or deleting users. Updating a hash uses the exact same logic as adding; however, deleting a user from the hash is slightly different.

TYPE THIS INTO PERL

```

if ( $cmd eq "delete" ) {
    if ( ! $db->{ 'users' }->{ $username } ) {
        print "Error: $username not found!\n";
    } else {
        delete( $db->{ 'users' }->{ $username } );
        lock_store( $db, userfile );
    }
}

```



RESULT

The system first checks if `$username` exists. If it does, the record is removed from the `$db->{ 'users' }` hash and the database is written to disk.

Check for Session Authorization (Step 1)

Every time a user accesses a restricted CGI script that imports your Authentication module, the first thing it does is to validate whether the user's session has been previously authorized. It does this by searching for the user's session identifier in the database, and matching it to a valid user record. If authorization is found, the script allows access to the original CGI. If authorization is not found, the script prompts the user for login credentials.

Checking for session authorization is the first of four steps required in order to allow a user access to a restricted CGI page with your Authentication module. On each page load, the CGI script first calls the module's `validate` method and waits for a return code indicating the results. If authorization passes, `validate` returns zero and the CGI script interprets this as a success: it executes its normal functionality. If authorization fails,

the Authentication module workflow continues to the second step: displaying the login prompt. If you are just starting to code your own Authentication module, this subroutine always produces a failing return code. This happens until you code the fourth and final step, authorizing a user's session, which produces the authorization grant that this subroutine is looking for.

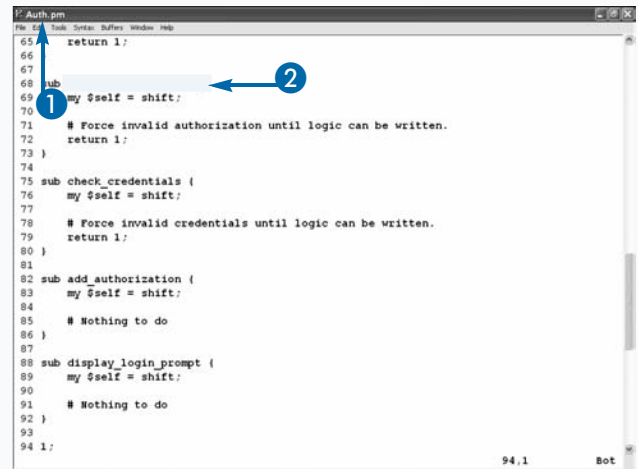
Checking for session authorization has an important secondary purpose. The subroutine must identify if an expired session is used by a visiting user, and force the user to re-authenticate by comparing the session's last-seen timestamp that is stored in the database against the current time. If the difference is too great, the subroutine displays the login prompt. Likewise, each time a valid session identifier is used with your Authentication module, the last-seen timestamp must be reset to the current time within the module's database.

Check for Session Authorization (Step 1)

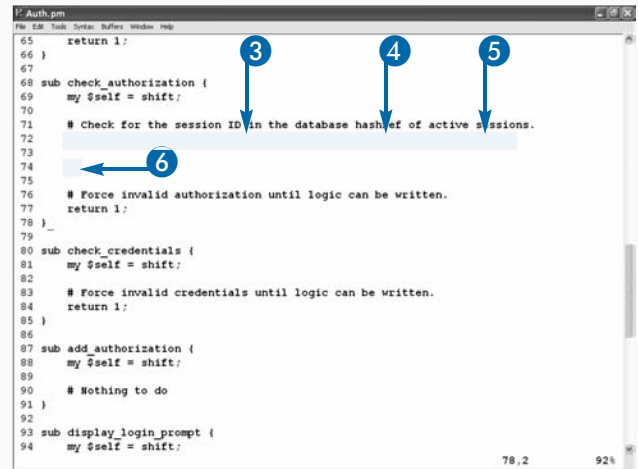
- 1 Open the Perl module `Auth.pm` in a text editor.
- 2 Scroll to the `check_authorization` subroutine.

Note: This subroutine, and the rest of the Authentication module's framework, was created in the section, "Create a Perl Authentication Module," earlier in this chapter.

- 3 Type `if ($self->{'db'}->{'sessions'}->{` to begin a conditional test that checks for a session in the database.
- 4 Type `$self->{'sessID'}` to specify the user's session identifier.
- 5 Type `}) {` to complete the conditional statement and open the conditional block.
- 6 Type `}` to close the conditional block.



```
Auth.pm
65 return 1;
66
67
68 sub
69 my $self = shift;
70
71 # Force invalid authorization until logic can be written.
72 return 1;
73 }
74
75 sub check_credentials {
76 my $self = shift;
77
78 # Force invalid credentials until logic can be written.
79 return 1;
80 }
81
82 sub add_authorization {
83 my $self = shift;
84
85 # Nothing to do
86 }
87
88 sub display_login_prompt {
89 my $self = shift;
90
91 # Nothing to do
92 }
93
94 1;
```



```
Auth.pm
65 return 1;
66
67
68 sub check_authorization {
69 my $self = shift;
70
71 # Check for the session ID in the database hashref of active sessions.
72
73
74
75
76 # Force invalid authorization until logic can be written.
77 return 1;
78 }
79
80 sub check_credentials {
81 my $self = shift;
82
83 # Force invalid credentials until logic can be written.
84 return 1;
85 }
86
87 sub add_authorization {
88 my $self = shift;
89
90 # Nothing to do
91 }
92
93 sub display_login_prompt {
94 my $self = shift;
```


- 7 Type **return 0**; to return success.
- 8 Update the comments to accurately reflect the return codes' results.
- 9 Type **my \$session =** before the conditional test expression.

Note: This makes accessing the correct session in memory easier within the conditional block.

- 10 Type **my \$timeout = 900**; to define an inactivity timeout in seconds.

- 11 Check if the difference between the current time and \$session->{ 'last-seen' } is within \$timeout.

- 12 Update the last-seen timestamp to the current time and write it to the database.

Note: Because \$session is a reference derived from \$self->{ 'db' }, updating the shortcut automatically updates the original hash ref.

- 13 Type **\$self->{ 'activeSession' } = \$session**;
- 14 Move the *success* return code within the timeout conditional block.
- 15 Save the Authentication module.

```

F:\Auth.pm
File Edit Tools Syntax Buffers Window Help
65 return 1;
66 }
67
68 sub check_authentication {
69     my $self = shift;
70
71     # Check for the session ID in the database hashref of active sessions.
72     if ( $self->{ 'db' }->{ 'sessions' }->{ $self->{ 'sessID' } } ) {
73         # 15 minute inactivity timeout.
74     }
75
76     # User authorization check failed, return a failure code.
77     return 1;
78 }
79
80 sub check_credentials {
81     my $self = shift;
82
83     # Force invalid credentials until logic can be written.
84     return 1;
85 }
86
87 sub add_authorization {
88     my $self = shift;
89
90     # Nothing to do
91 }
92
82,2 87%
  
```

```

F:\Auth.pm
File Edit Tools Syntax Buffers Window Help
63 # Instruct the calling Perl CGI to not run normally.
64 return 1;
65 }
66
67 sub check_authentication {
68     my $self = shift;
69
70     # Check for the session ID in the database hashref of active sessions.
71     if ( $session = $self->{ 'db' }->{ 'sessions' }->{ $self->{ 'sessID' } } ) {
72         my $timeout = 900; # 15 minute inactivity timeout.
73
74         if (
75             # Update the database with the new last-seen timestamp
76             # Link the authorized session into module itself
77             # User authorization check passed, return a success code.
78         ) {
79             # User authorization check failed, return a failure code.
80             return 1;
81         }
82     }
83 }
84
85
86
87
88
89
90
91
92
67,5 68%
  
```

Apply It

Deciding on the timeout depends on the level of security of the information on your site. Because high-security Web sites, like those that offer online banking services, do not want unauthorized users to access information left on an idle browser by an absent account owner, an unusually low timeout of two or three minutes is common.

The `time` function returns the number of seconds since January 1, 1970. Because it is already in a numeric form, it is very easy to use it for calculating time differences, such as in your inactivity-timeout test.

If you are interested, you can convert a timestamp into a readable string with the help of the `localtime` function. This can be useful when logging the literal date and time the user last visited the Web site.

TYPE THIS

```
print localtime( SECONDS );
```



RESULTS

```
Mon Apr 2, 03:21:56 2010
```

You can customize the output format of `localtime` by assigning it into an array instead of a scalar. For more information, run the command-line PerlDoc program, `perldoc -f localtime`.

Display a Login Prompt (Step 2)

After confirming that a user's session identifier lacks proper authorization in the database, the next thing to do is present the user with a prompt to enter their credentials. This user-login prompt supersedes the normal output of every Perl CGI script that imports your Authentication module. This *display a login prompt* step is the second of four steps required in order to allow a user access to a restricted CGI page with your Authentication module. The login prompt appears to the user when the Authentication module identifies that the user's session ID lacks authorization to view the current Web page. The login prompt appears instead of the regular CGI script's output; the Web page URL the user is on remains the same.

The actual HTML code for the prompt should be displayed using the `HTML::Template` module. This makes it easier

for you to change the HTML simply by editing the template, and to include unique data within the template.

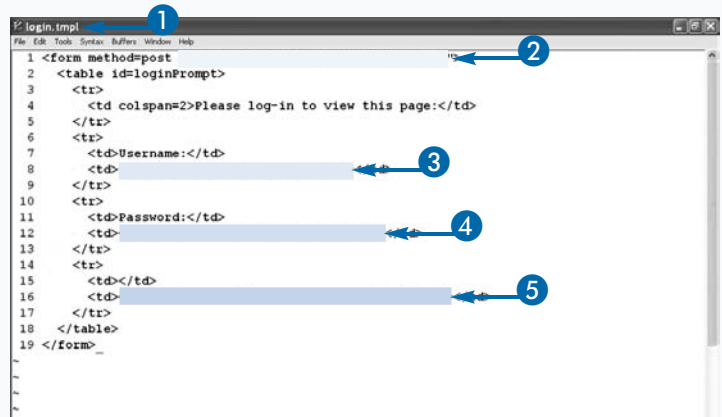
When `validate` needs to call the `display_login_prompt` subroutine, the user is not currently authenticated; therefore you need to send a signal to the main CGI script, stopping it from producing its normal CGI output. This happens when `validate` returns a failure code as a return value. When the calling script receives this signal, it immediately stops and displays nothing to the user. Therefore, the `display_login_prompt` must produce the HTTP header and display the login prompt HTML.

After completing this section, proceed to "Validate a User's Credentials," which will accept the data from the login prompt and continue the validation process. If it fails, it will pass control back to this login prompt and display an appropriate error message to the user.

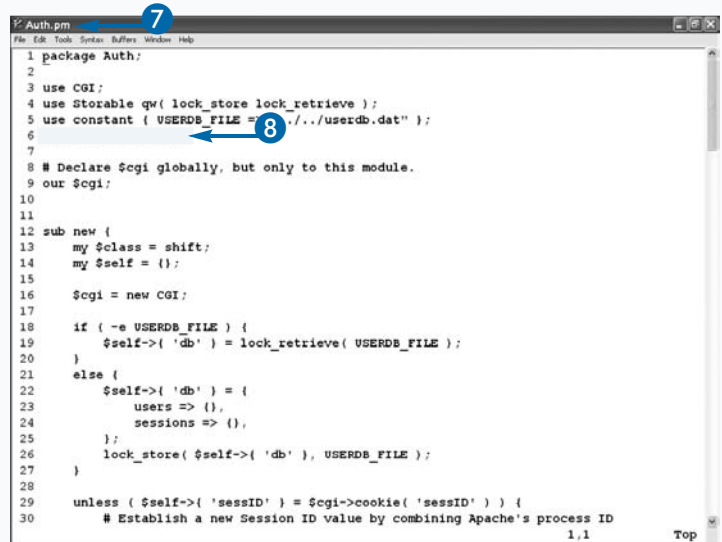
Display a Login Prompt (Step 2)

- 1 Create a new login template.
- 2 Type `action=<tmpl_var name=SCRIPT_NAME>` into the `<form>` tag.
- 3 Type `<input type=text name=username>` to create a username prompt.
- 4 Type `<input type=password name=password>` to create a password prompt.
- 5 Type `<input type=submit name=process value=Login>` to create a login button.
- 6 Save the template file as `login.tmpl`.
- 7 Open the Perl module `Auth.pm` in a text editor.
- 8 Type `use HTML::Template;` to import the module.

Note: *HTML::Template* is a third-party module that is not part of the standard Perl installation. See Chapter 13 for information on using *HTML::Template*.



```
1 <form method=post
2 <table id=loginPrompt>
3 <tr>
4 <td colspan=2>Please log-in to view this page:</td>
5 </tr>
6 <tr>
7 <td>Username:</td>
8 <td>
9 </tr>
10 <tr>
11 <td>Password:</td>
12 <td>
13 </tr>
14 <tr>
15 <td colspan=2>
16 <td>
17 </tr>
18 </table>
19 </form>
```



```
1 package Auth;
2
3 use CGI;
4 use Storable qw( lock_store lock_retrieve );
5 use constant { USERDB_FILE => ../../userdb.dat };
6
7
8 # Declare $cgi globally, but only to this module.
9 our $cgi;
10
11
12 sub new {
13 my $class = shift;
14 my $self = {};
15
16 $cgi = new CGI;
17
18 if ( -e USERDB_FILE ) {
19 $self->{ 'db' } = lock_retrieve( USERDB_FILE );
20 }
21 else {
22 $self->{ 'db' } = {
23 users => {},
24 sessions => {},
25 };
26 lock_store( $self->{ 'db' }, USERDB_FILE );
27 }
28
29 unless ( $self->{ 'sessID' } = $cgi->cookie( 'sessID' ) ) {
30 # Establish a new Session ID value by combining Apache's process ID
31 1,1
32 }
```

- 9 Scroll to the `display_login_prompt` subroutine.
- 10 Initialize the `HTML::Template` module.
- 11 Use `login.tmpl` as the template.

```

102 # are not properly defined.
103 return 1 if ( ! $username || ! $password || ! $userdb );
104
105 # Force invalid credentials until logic can be written.
106 return 1;
107 }
108
109 sub add_authorization {
110     my $self = shift;
111
112     # Nothing to do
113 }
114
115 sub
116     my $self = shift;
117
118     my $tmpl =
119         filename =>
120         die_on_bad_params => 0,
121     };
122 }
123
124 1;
  
```

- 12 Type `$tmpl->param(SCRIPT_NAME => $ENV{'SCRIPT_NAME' });`.

Note: The template's form uses `SCRIPT_NAME` so it knows where to submit the username and password. CGI environment variables are not automatically inherited by HTML templates, so you need to manually add them as a template parameter.

- 13 Print the HTTP header.
- 14 Print the template output.

```

102 # are not properly defined.
103 return 1 if ( ! $username || ! $password || ! $userdb );
104
105 # Force invalid credentials until logic can be written.
106 return 1;
107 }
108
109 sub add_authorization {
110     my $self = shift;
111
112     # Nothing to do
113 }
114
115 sub display_login_prompt {
116     my $self = shift;
117
118     my $tmpl = HTML::Template->new(
119         filename => "login.tmpl",
120         die_on_bad_params => 0,
121     );
122
123     # Forward the SCRIPT_NAME environment from CGI to the template
124
125
126     # Display the login prompt
127
128 }
129
130
131 1;
  
```

Extra

It is a good idea to display an appropriate message to the user that reflects what state they are in: new login attempt, failed login attempt (try again), or timed-out session (try again). Right now you are hard-coding the login message to assume it is always a new login attempt. You can customize the `login.tmpl` file, add a message parameter, and then code `display_login_prompt` to populate the parameter.

Another feature that you can add is a password recovery page. While developing such a feature is beyond the scope of this book, your users will appreciate the added convenience. Your password recovery page should be a dedicated CGI script that, after using a form of *alternative identification*, resets the user's password. The alternative identification means you need to confirm that the user requesting the password recovery is the actual account holder, without using the original password. Often, you can do this by sending an e-mail to the user, using an address already stored on file. Once received, the e-mail can contain a link with a random token that, when clicked, resets the user's password. Only the original account holder would have access to that e-mail message, assuming the e-mail address is valid.

Validate a User's Credentials (Step 3)

After the user submits their username and password with the login prompt, the Authentication module must validate the user's credentials against the database. Only after the credentials are correctly matched is the user's session granted access. Otherwise, the login prompt appears again, with an error message stating that the previous authentication attempt had failed.

This *validate the user's credentials* step is the third of four steps required to enable a user to access a restricted CGI page with your Authentication module. In order to validate the credentials, you need to receive the username and password input from the HTML using CGI parameters. The username and password values need to be properly defined, and a user profile record needs to exist in the database. It is strongly recommended that

Validate a User's Credentials (Step 3)

- 1 Type `use Digest::SHA qw(sha1_base64)`; to import the SHA module's `sha1_base64` function.

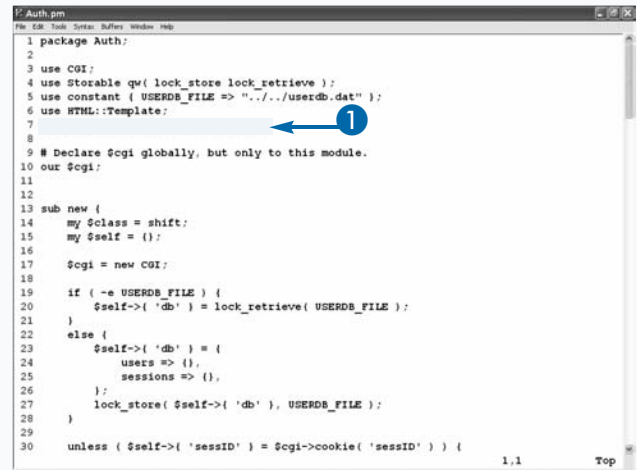
Note: The password hash format here must match the format used in the section, "Store User Credentials in a User's Database."

- 2 Scroll to the `validate` function.

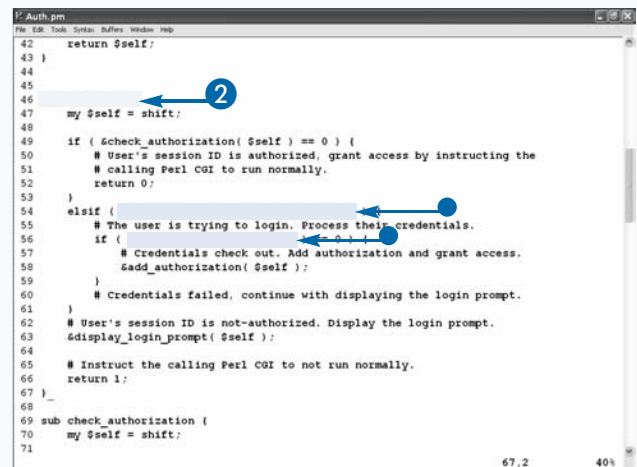
- The `check_credentials` function is called only when the login form submits `process=Login`, as defined by the button in `login.tmpl`.

you activate some sort of Web site encryption, such as TLS/SSL in Apache, before you collect the user's credentials. While TLS/SSL encryption is not required after the credentials have been validated, it gives the user the confidence that your Web site is safe and secure while they are logged in.

The validation process involves hashing the CGI password with the same algorithm used earlier in this chapter, Digest::SHA, and comparing this value with the already-hashed database password. If they match, the user is allowed to proceed. If they do not match, an appropriate error message appears above the login prompt. The actual error message needs to appear within the login prompt template, but only in the event of a previous failed attempt. When the user's credentials are validated, the database updates and activates the user's session identifier, which actually grants access.



```
1 package Auth;
2
3 use CGI;
4 use Storable qw( lock_store lock_retrieve );
5 use constant ( USERDB_FILE => "../userdb.dat" );
6 use HTML::Template;
7
8
9 # Declare $cgi globally, but only to this module.
10 our $cgi;
11
12
13 sub new {
14     my $class = shift;
15     my $self = {};
16
17     $cgi = new CGI;
18
19     if ( !e USERDB_FILE ) {
20         $self->{ 'db' } = lock_retrieve( USERDB_FILE );
21     }
22     else {
23         $self->{ 'db' } = {
24             users => {},
25             sessions => {},
26         };
27         lock_store( $self->{ 'db' }, USERDB_FILE );
28     }
29
30     unless ( $self->{ 'sessID' } = $cgi->cookie( 'sessID' ) ) {
```



```
42     return $self;
43 }
44
45
46 my $self = shift;
47
48 if ( $check_authorization( $self ) == 0 ) {
49     # User's session ID is authorized, grant access by instructing the
50     # calling Perl CGI to run normally.
51     return 0;
52 }
53
54 elsif (
55     # The user is trying to login. Process their credentials.
56     if (
57         # Credentials check out. Add authorization and grant access.
58         $add_authorization( $self );
59     )
60     # Credentials failed, continue with displaying the login prompt.
61 ) {
62     # User's session ID is not-authorized. Display the login prompt.
63     $display_login_prompt( $self );
64
65     # Instruct the calling Perl CGI to not run normally.
66     return 1;
67 }
68
69 sub check_authorization {
70     my $self = shift;
71 }
```

- 3 Scroll to the `check_credentials` function.
- 4 Store the CGI's username and password parameters as new variables.
- 5 Create a shortcut scalar to the user's profile in the database.
- 6 Type `return 1 if (! $username || ! $password || ! $userdb);` to ensure that username, password, and database are all properly defined before continuing.

Note: Typing a conditional statement like this is equivalent to using a conditional block containing the single command.

- 7 Type `return 0 if (sha1_base64($password) eq $userdb->{ 'pwhash' });` to return success if the submitted and database hashes match.
- 8 Update the final comment, in case the password hashes do not match.

```

F:\Auth.pm
File Edit Tools Settings Buffers Window Help
86      # User authorization check passed, return a success code.
87      return 0;
88  }
89  }
90
91  # User authorization check failed, return a failure code.
92  return 1;
93 }
94
95 sub ← 3
96     my $self = shift; ← 4
97
98     ← 5
99
100
101
102     # Return a failure code if either the username, password, or database profile
103     # are not properly defined. ← 6
104
105
106     # Force invalid credentials until logic can be written.
107     return 1;
108 }
109
110 sub add_authorization {
111     my $self = shift;
112
113     # Nothing to do
114 }
115

```

```

F:\Auth.pm
File Edit Tools Settings Buffers Window Help
86      # User authorization check passed, return a success code.
87      return 0;
88  }
89  }
90
91  # User authorization check failed, return a failure code.
92  return 1;
93 }
94
95 sub check_credentials {
96     my $self = shift;
97
98     my $username = $cgi->param( 'username' );
99     my $password = $cgi->param( 'password' );
100     my $userdb = $self->{ 'db' }->{ 'users' }->{ $username };
101
102     # Return a failure code if either the username, password, or database profile
103     # are not properly defined.
104     return 1 if ( ! $username || ! $password || ! $userdb );
105
106     # Compare the hashed login password with the previously hashed database password
107     return 0 if ← 8
108
109
110     return 1;
111 }
112
113 sub add_authorization {
114     my $self = shift;
115

```

Extra

By hashing your user's passwords, you are making it much more difficult for an attacker to access the user's original password value. A hash, by definition, has four primary features that make it ideal for passwords. First, it is very easy to take source data and create a cryptographic hash; second, it is very difficult to derive the original data from the hash; third, it is impossible to modify the source data without changing the calculated hash; fourth, it is extremely unlikely that you will find two sources that generate the same hash.

Earlier in this chapter, you used the `Digest::SHA` module to produce a SHA-1 hash of the original password and write it to the database. Therefore, as you have seen here, you must use the same module to rehash the submitted password when the user authenticates. The hashed values must match, thus indicating that the user knows what the original password value is.

Perl contains built-in support for more complex hashing algorithms. To compare, SHA-1 produces output of 160 bits, which is considered minimally secure. You have the option to use SHA-256, SHA-384, and SHA-512 algorithms to increase the mathematical complexity of your SHA password-hashes. For more information, check the `Digest` module's `Perldoc` page.

Authorize a User's Session (Step 4)

Assuming that the user has correctly provided their username and password, the Authentication module completes its validation process by granting authorization to the user's session. It does this by linking the user's session identifier to their user profile in the database, and writing a timestamp.

This *authorize a user's session* step is the final step required to enable user access to a restricted CGI page with your Authentication module. This grant is sufficient to allow the user persistent access to any restricted CGI script, without needing to revalidate their credentials on each Web page click. The actual grant is stored in the database by linking the *session* hashref with the *user* hashref, and then assigning a starting value to the 'last-seen' timestamp.

Authorize a User's Session (Step 4)

- 1 Scroll to the `validate` subroutine.
- 2 Type `print "Location: $ENV{'HTTP_REFERER'}\n\n";` to redirect the user's Web browser.
- 3 Change the return code from 0 to 1.

Note: The previous *Location* redirection will send the user back to `check_authorization`; the return 1 code inhibits normal CGI output, at least until after the redirection.

- 4 Scroll to the `add_authorization` subroutine.
- 5 Type `$self->{'db'}->{'sessions'}->{$self->{'sessID'}} = {` to begin the update session statement.
- 6 Type `};` to close the statement.

The Authentication module passes control to the top of the workflow process by redirecting the user's Web browser back to the URL in the `HTTP_REFERER` environment value, or in other words, the URL that the user came from when the login prompt appeared. This redirection is extremely important as it prevents the username and password fields from being present at any point when the CGI script is running normally. If you used the `GET` method in the login template, you would see the username and password in the URL. This is mitigated with `POST`, but the browser is still aware of the CGI fields.

Another benefit of the redirection is that bookmarks now work correctly. For example, if the user were to bookmark `http://mydomain.com/secure_script.pl`, you would want the bookmark to work regardless of whether the user is in an authorized or unauthorized state.

```
1  sub
2      my $self = shift;
3
4      if ( $check_authorization( $self ) == 0 ) {
5          # User's session ID is authorized, grant access by instructing the
6          # calling Perl CGI to run normally.
7          return 0;
8      }
9
10     elsif ( $cgi->param( "process" ) eq "Login" ) {
11         # The user is trying to login. Process their credentials.
12         if ( $check_credentials( $self ) == 0 ) {
13             # Credentials check out. Add authorization and grant access.
14             $add_authorization( $self );
15         }
16
17         # Redirect the user back to the original URL. This removes any
18         # stale CGI parameters lagging around the submitted login URL or in
19         # the browser cache.
20
21         # Instruct the calling Perl CGI not to run normally, quite yet. The
22         # previous redirection will direct the user back to
23         # $check_authorization() via HTTP_REFERER. At this
24         # point, access will be normally granted if the grant exists.
25     }
26
27     # User's session ID is not-authorized. Display the login prompt.
28     $display_login_prompt( $self );
29
30     # Instruct the calling Perl CGI to not run normally.
```

```
116 # Compare the hashed login password with the previously hashed database password.
117 return 0 if ( sha1_base64( $password ) eq $userdb->{'pwhash' } );
118
119 # Return failure code, password hashes do not match.
120 return 1;
121 }
122
123 sub
124     my $self = shift;
125
126     # Assign the authorization grant between the user's session and profile
127
128
129
130 }
131
132 sub display_login_prompt {
133     my $self = shift;
134
135     my $tmpl = HTML::Template->new(
136         filename => "login.tmpl",
137         die_on_bad_params => 0,
138     );
139
140     # Forward the SCRIPT_NAME environment from CGI to the template
141     $tmpl->param( SCRIPT_NAME => $ENV{'SCRIPT_NAME' } );
142
143     # Display the login prompt
144     print $cgi->header();
145     print $tmpl->output;
```


- 7 Type `'username' => $cgi->param('username')`, to store the username in the session.
- 8 Type `'profile' => $self->{ 'db' }->{ 'users' }->{ $cgi->param('username') }`, to link the profile to the session.

Note: Here you are linking the username separate from the profile, because in the profile hash ref there is no 'username' key.

- 9 Type `'last-seen' => time`, to create the first timestamp in the session.
- 10 Write the authorization grant to the database.
- 11 Save the `Auth.pm` Perl module.

```

116 # Compare the hashed login password with the previously hashed database password.
117 return 0 if ( sha1_base64( $password ) eq $userdb->{ 'pwhash' } );
118
119 # Return failure code, password hashes do not match.
120 return 1;
121 }
122
123 sub add_authorization {
124     my $self = shift;
125
126     # Assign the authorization grant between the user's session and profile
127     $self->{ 'db' }->{ 'sessions' }->{ $self->{ 'sessID' } } = {
128         'username' => $cgi->param( 'username' ),
129         'profile' => $self->{ 'db' }->{ 'users' }->{ $cgi->param( 'username' ) },
130     };
131 }
132
133 sub display_login_prompt {
134     my $self = shift;
135
136     my $tmpl = HTML::Template->new(
137         filename => "login.tmpl",
138         die_on_bad_params => 0,
139     );
140
141     # Forward the SCRIPT_NAME environment from CGI to the template
142     $tmpl->param( SCRIPT_NAME => $ENV{ 'SCRIPT_NAME' } );
143
144     # Display the login prompt
145     print $cgi->header();

```

```

116 # Compare the hashed login password with the previously hashed database password.
117 return 0 if ( sha1_base64( $password ) eq $userdb->{ 'pwhash' } );
118
119 # Return failure code, password hashes do not match.
120 return 1;
121 }
122
123 sub add_authorization {
124     my $self = shift;
125
126     # Assign the authorization grant between the user's session and profile
127     $self->{ 'db' }->{ 'sessions' }->{ $self->{ 'sessID' } } = {
128         'username' => $cgi->param( 'username' ),
129         'profile' => $self->{ 'db' }->{ 'users' }->{ $cgi->param( 'username' ) },
130     };
131
132     # Write the authorization grant to the database
133
134 }
135
136
137 sub display_login_prompt {
138     my $self = shift;
139
140     my $tmpl = HTML::Template->new(
141         filename => "login.tmpl",
142         die_on_bad_params => 0,
143     );
144
145     # Forward the SCRIPT_NAME environment from CGI to the template

```

Extra

Right now, for every user that you authorize, there is currently no database cleanup. This means that data will continue to be added to `$self->{ 'db' }->{ 'sessions' }`, but sessions that have expired a long time ago will never be deleted. An easy way around this problem is to write a little janitor Perl script that reads all session entries, compares the last-seen key to the current time, and deletes it from the database.

Technically speaking, you can add the same functionality to `check_authorization`. If it finds a session that has expired, delete it from the database. However, this would only purge expired sessions when user actually attempts to authenticate with that session. Remember that when a user closes their browser, their session identifier cookie is automatically deleted. When they revisit the site, a new cookie will be assigned and the previous session will remain in the database.

Again, the best way to solve this problem is to write a janitor script that runs every hour or so, which actively checks for stale sessions by comparing last-seen timestamps, and cleans them from the database.

Restrict Access to a CGI Script

You can use your completed Authentication module to restrict access to any CGI script, now that the module supports all four stages of the validation workflow, as described throughout this chapter.

To actually apply the module to an existing CGI script is extremely easy. There are three simple changes that you need to apply at the top of your CGI script. Once complete, the authorization procedure happens every time a user accesses your script in a Web browser. Depending on the state of their session, the Authentication module determines what the browser should display: a login prompt or the normal CGI script.

The first thing to do is to import your new Authentication module into a CGI script that will require authentication:

```
use Auth.pm;
```

Second, once the module is loaded, you need to initialize the module into a reference handle variable. All methods and variables controlled by your Authentication module will be accessible from this variable:

```
my $auth = new Auth;
```

Third, you call the validation code prior to any native functionality built-in to the restricted CGI script:

```
$auth->validate && exit 1;
```

The last half of this statement is new. The `validate` method returns 0 when the user is correctly logged in, and 1 if they are not. Because the Authentication module contains the ability to display a login prompt, you need to avoid the restricted CGI script's normal output. So, if `validate` returns 1, the `&& exit 1` code automatically activates. If `validate` returns 0, the `exit` is ignored and the restricted CGI script continues normally.

Restrict Access to a CGI Script

- 1 Open a Perl CGI script in a text editor.
- 2 Type `use Auth;` to import the completed Authentication module.
- 3 Type `my $auth = new Auth;` to initialize the module.
- 4 Type `$auth->validate() && exit 1;` to validate the user.
- 5 Save the script.

```
1 #!C:\Perl\bin\perl.exe
2
3 use HTML::Template;
4 use CGI;
5
6
7 my $tmpl = HTML::Template->new(
8     filename => "restricted.tpl",
9     die_on_bad_params => 0,
10 );
11 my $cgi = new CGI;
12
13
14
15
16 # Relay the active-session to the template
17 $tmpl->param( $auth->{ 'activeSession' } );
18
19 # Begin normal CGI functionality
20 print $cgi->header;
21 print $tmpl->output;
```

Annotations: 1 points to the shebang line, 2 points to the `use HTML::Template;` line, 3 points to the `my $cgi = new CGI;` line, 4 points to the blank line after `new CGI;`, and 5 points to the blank line before the comment on line 16.

- 6 Open the restricted CGI script's template.
- 7 Type `<tmpl_var name="username">` to display the logged-in user's name.

Note: Anything that is found under `$db->{ 'activeSession' }` is now accessible from the template.

- 8 Save the restricted template file.

```
1 <h3>Restricted CGI script</h3>
2 Thank you for logging on,
3
4
5
6
7
```

Annotations: 6 points to the blank line after the first line, and 7 points to the `<tmpl_var name="username">` line.

- 9 Load the Perl CGI script in a Web browser.

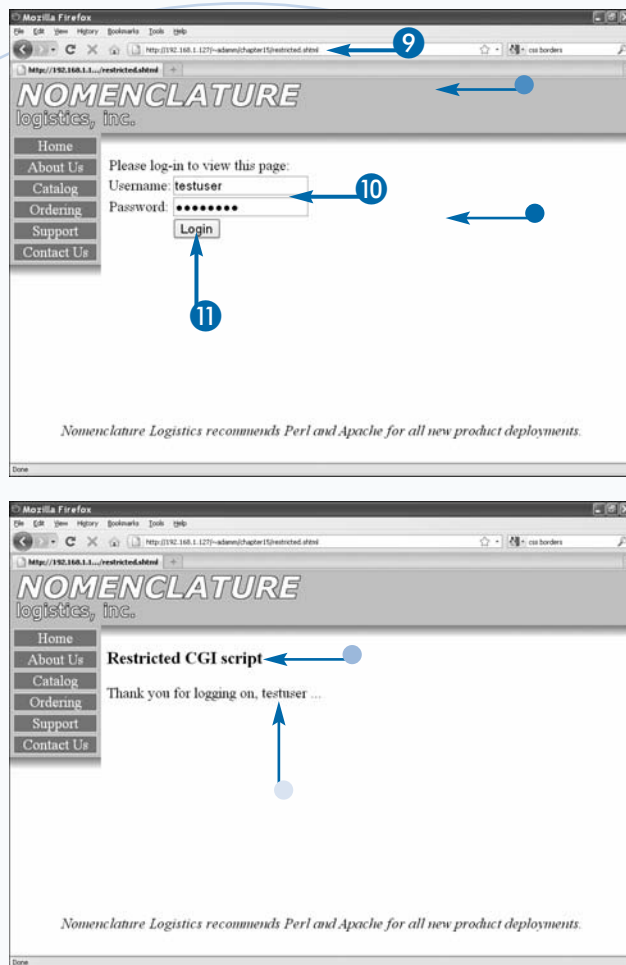
Note: If the Perl CGI script has already been linked into SHTML, as described in Chapter 14, load the SHTML file instead.

- The login prompt appears.
 - The browser displays the SHTML header and footer, if applicable.
- 10 Type in the username and password for a user in the database.
 - 11 Click Login.

The page reloads.

- The browser displays the restricted CGI script output.
- The username appears.

Note: If you close your browser, the session cookie is deleted. Opening the page again redisplay the login prompt as your new session cookie is a different identifier value.



Apply It

One issue with this Authentication module implementation is that it cannot easily restrict access to a static file. If you want to restrict an HTML file, it is possible to do this by reading the target file as a HTML::Template input template, and passing it through a restricted Perl CGI script.

To restrict a text or image file, first ensure that all restricted files are stored in a directory that is not accessible by a normal URL. Second, replace the `$cgi->header()` and `HTML::Template` components with something that can access the original data with the correct MIME type.

For example, if you want to restrict JPEG images, you can create a `restrict-jpeg.pl` script that uses the Authentication module, as well as the following commands:

```
print "content-type: image/jpeg\n\n";
open ( DATA, $cgi->param( 'file' ) );
while ( <DATA> ) { print $_; }
close( DATA );
```

Direct users to the Web page `http://mydomain.com/restrict-jpeg.pl?file=image0001.jpg`; on the first load they are prompted for a username and password. On subsequent loads, they see the original image.

```
print "Location: $ENV{ 'HTTP_REFERER' }\n\n";
```

Using this redirector creates the illusion that the user never left the restricted page; the actual URL never

- 1 Open the Perl module `Auth.pm` in a text editor.
- 2 Create a new method called `logout`.
- 3 Type `delete($self->{ 'db' }->{ 'sessions' }->{ $self->{ 'sessID' } });` to delete the session.
- 4 Write the database memory to disk.

This is the final step to securing CGI pages with your own Authentication module. If you had implemented Apache authentication, you would find that Apache does not provide an equivalent logoff feature. Only by controlling the entire authentication process can you add this feature into your Web site.

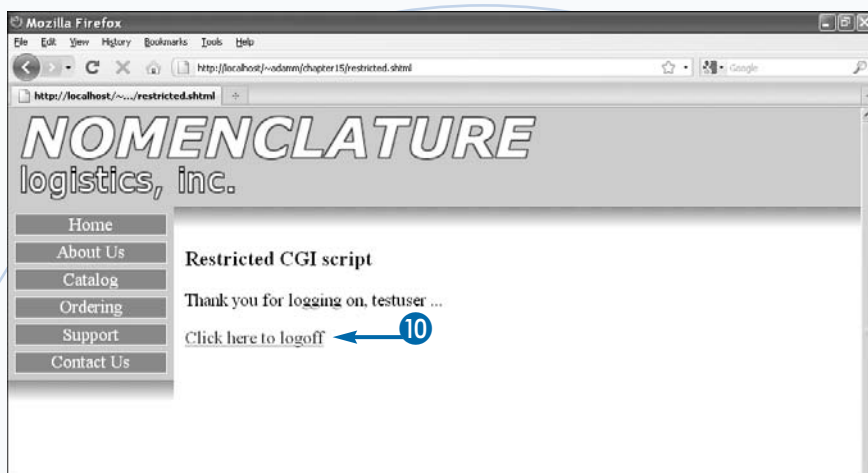


Note: Add a HREF link to `logout.pl` in all of your restricted-access templates.

- 9 Go to a restricted Web page where you are currently logged in.

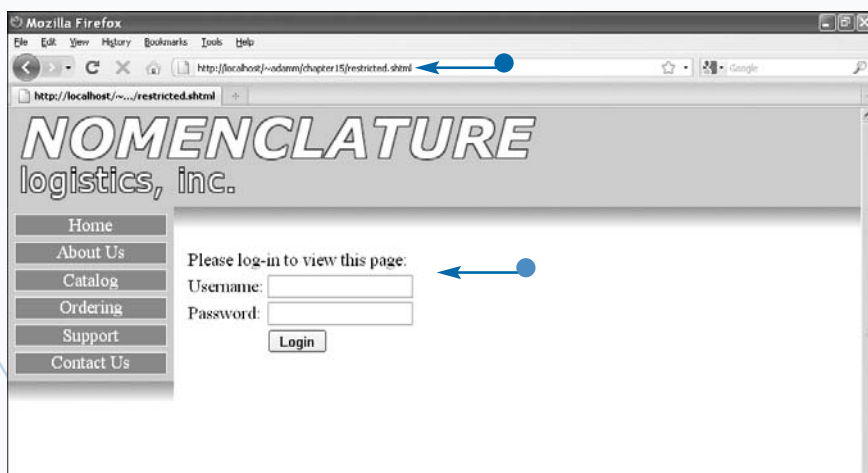
The Web page loads.

- 10 Click the link to your `logout.pl` script.



The Web page reloads.

- The Web page URL stays the same.
- The login prompt appears.



Apply It

Displaying a *You have been logged-off* message is actually quite complex. You want the process to be initiated in the `logout.pl` script, but to appear within a restricted CGI script's login prompt. You can implement this process by saving the message in the users' database, attached to the original session identifier, but disconnected from the user profile. This may seem unusual, but it is the only way to transfer information from one CGI to another.

The first step, storing the message, can happen anywhere under `$db` outside of the users and sessions hash refs. You can use this command just prior to writing `$db` to disk:

```
$db->{ 'message' }->{ $sessID } = "You have been logged off";
```

Next, `Auth.pm`'s `display_prompt` requires something like this:

```
$tmpl->param( message => $self->{ 'db' }->{ 'message' }->{
    $self->{ 'sessID' } } );
delete( $self->{ 'db' }->{ 'message' }->{ $self->{ 'sessID' } } );
```

You need the `delete` here because you only want the message to appear once. As described in Chapter 6, `delete` can be used to trim a specific key from a hash.

Finally, add `<tmpl_var name=message>` into `login.tmpl` to display your new "logged-off" message as HTML output.

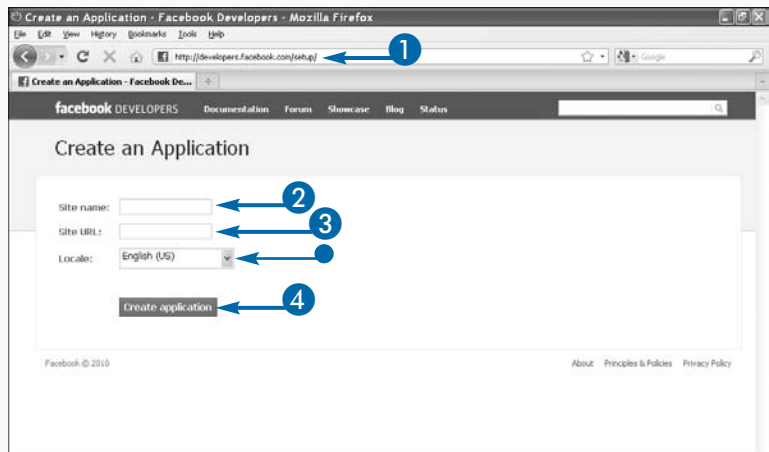
Register Your Web Site as a Facebook Application

Registering as a Facebook Application grants you access to the Facebook JavaScript SDK, which you can use to add some cool Facebook features directly to your Web site. Registration also grants you access to create a Facebook Application with a Perl CGI script. Once registered, you are assigned an Application ID, an API Key, and an Application Secret. Together, these values are used by your Web site to access the Facebook servers.

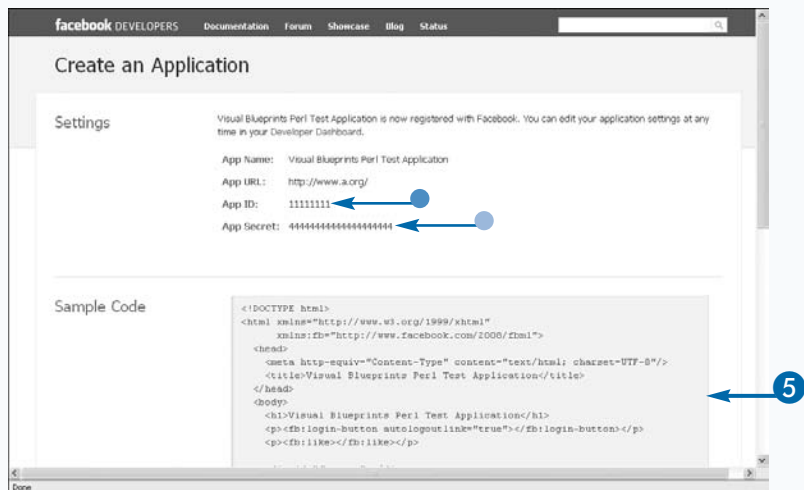
You can use the Facebook JavaScript SDK to install simple Facebook features, such as Social plugins and single sign-on authentication. Because JavaScript only executes on the Web browser, your Web server cannot monitor what end-users are doing with Facebook. The JavaScript SDK is the first thing you set up once you register your Web site as a Facebook Application.

Register Your Web Site as a Facebook Application

- 1 Go to the URL <http://developers.facebook.com/setup/>.
- 2 Type in your Web site's name.
- 3 Type in your URL.
 - You can select an alternative locale, if required.
- 4 Click Create application.



- Your App ID.
 - Your App Secret code.
- 5 Copy the sample code into a buffer.



- 6 Open a text editor.
- 7 Paste the Facebook sample code.
- 8 Save the static HTML file on your Web site.

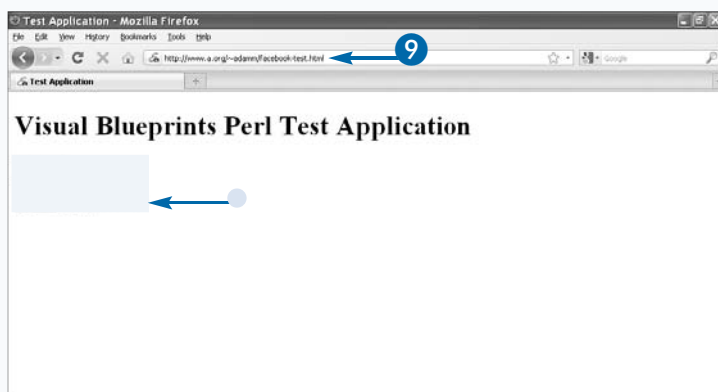
```

1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:fb="http://www.facebook.com/2008/fbml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
6 <title>Test Application</title>
7 </head>
8 <body>
9 <h1>Visual Blueprints Perl Test Application</h1>
10 <p><fb:login-button autologoutlink="true"></fb:login-button></p>
11 <p><fb:like></fb:like></p>
12
13 <div id="fb-root"></div>
14 <script>
15   window.fbAsyncInit = function() {
16     FB.init({appId: '1111111', status: true, cookie: true,
17           xfbml: true});
18   };
19   (function() {
20     var e = document.createElement('script');
21     e.type = 'text/javascript';
22     e.src = document.location.protocol +
23           '//connect.facebook.net/en_US/all.js';
24     e.async = true;
25     document.getElementById('fb-root').appendChild(e);
26   }());
27 </script>
28 </body>
29 </html>

```

- 9 Navigate to the static HTML Web page.

- This is an example of Facebook JavaScript SDK functionality, including Like and Login buttons.



Apply It

After you register your Web site, you can add the JavaScript SDK to any Web page. If you have configured SSI as described in Chapter 14, just add the SDK code into your header .shtml file. Just be sure to set APP_ID to your actual Application ID. Only the following code is required.

TYPE THIS

```

<div id="fb-root"></div>
<script>
  window.fbAsyncInit = function() {
    FB.init({appId: 'APP_ID', status: true,
      cookie: true, xfbml: true});
  };
  (function() {
    var e = document.createElement('script'); e.async =
true;
    e.src = document.location.protocol +
      '//connect.facebook.net/en_US/all.js';
    document.getElementById('fb-root').appendChild(e);
  }());
</script>

```

RESULTS

Any Web page with this code can now use the Facebook JavaScript SDK.

Add a Facebook Social Plugin to Your Web Site

You can add a Facebook Social plugin into your Web site, which allows your users to experience your Web site with their Facebook friends. There are several Facebook features available as Social plugins; the easiest to implement is a Like button that, when clicked by a visiting user, adds a message into their Facebook newsfeed that they like your Web site.

Because Facebook hosts the content, the plugin can display personalized content even if it is the first time a user visits your site. It is actually the Facebook JavaScript SDK that is providing the core feature, which means that you, as the Web site owner, cannot view what is being displayed on the user's Web browser.

Social plugins also display customized content on your Web site, related to the visiting user's Facebook profile.

As a result, only content related to that user, and their friends, is displayed. Occasionally, generic statistical information may also appear.

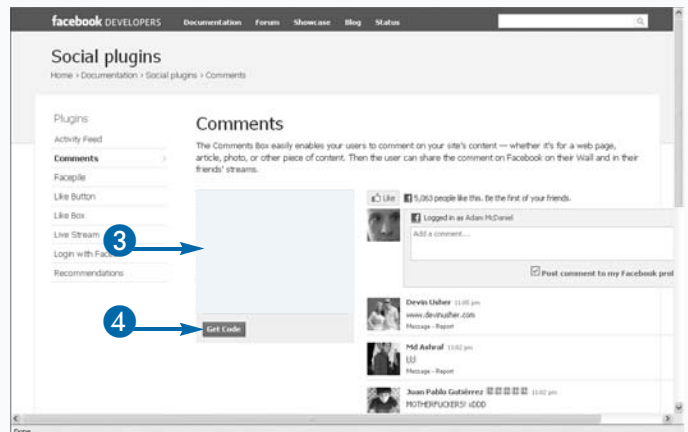
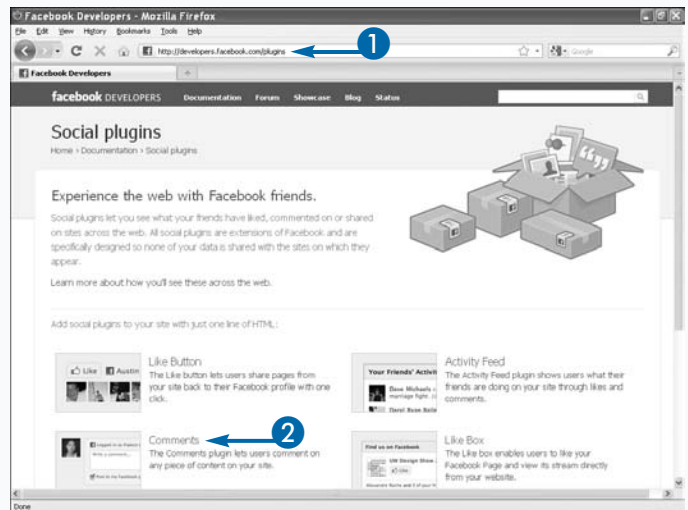
For example, in the case of a Like button, when you connect to your own Web site and click it, you may read, "You, Johnny Smith, Jane Doe, and 12 other people like this site." When a different person visits and clicks the button, she may read, "You, Mike Jackson, Chris Hall, and 12 other people like this site."

While this feature does not require any Perl- or Apache-specific code, it is the starting point to adding in some form of social connectivity to your Web site.

You can find the full list of available Facebook Social plugins online at <http://developers.facebook.com/plugins>.

Add a Facebook Social Plugin to Your Web Site

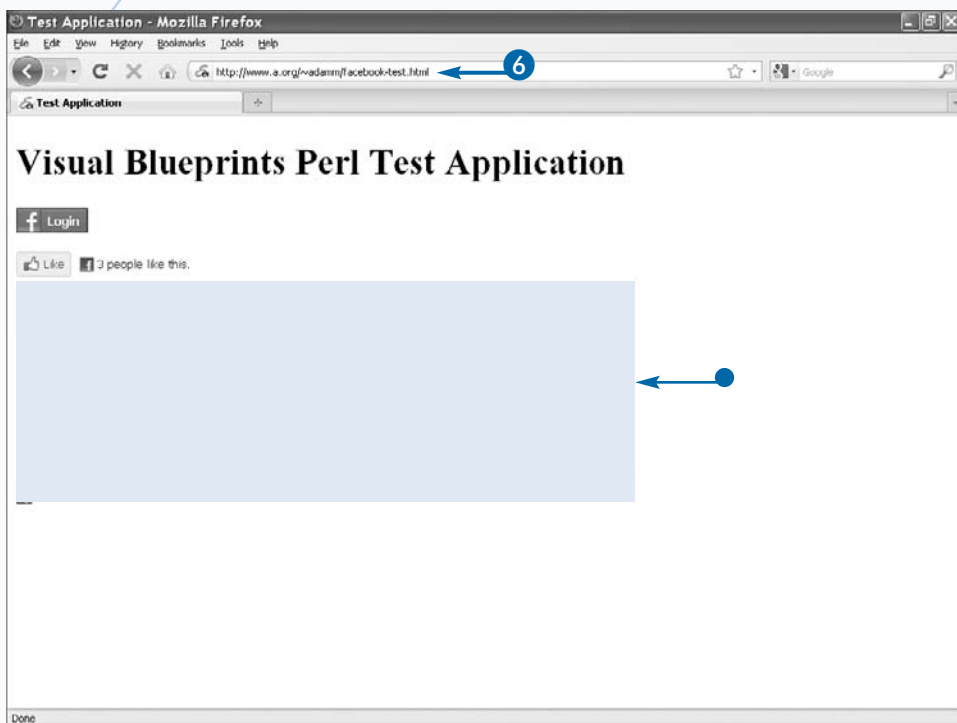
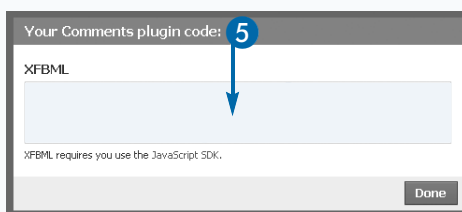
- 1 Go to the URL <http://developers.facebook.com/plugins>.
- 2 Click a plugin.
- 3 Customize the button by changing the fields.
- 4 Click Get Code.



- 5 Copy the XFBML code and paste it on your Web site.

Note: If you have not yet registered your Web site and are using the JavaScript SDK, then copy the iframe code example instead (if it is available).

- 6 View your Web site in a browser.
 - The Facebook Social plugin appears.



Extra

When you click the Like button, Facebook registers your vote for the site and links it to your profile for your friends to see. Other Social plugins are also available for you to implement, using the same basic procedure as the Like button.

The Comments plugin allows your users to post comments on any page of content on your site with their Facebook profile. If you are interested in running a blog, this plugin is the easiest way to implement some form of commenting feature onto individual topic pages.

The Activity Feed plugin shows other users what recent activity their friends have been doing on your Web site. This includes other Web pages that users like, along with posted comments. With this feed, you can provide a simple stream aggregating all activity on your Web site.

The Live Stream plugin lets users visiting your Web site share activity and comments in real time. The plugin appears in the form of a Live Stream Box, which works best when you are running a real-time event.

Enable Facebook Connect on Your Web Site

You can enable Facebook Connect, a single sign-on authentication service provided by Facebook, on your Web site using the Facebook JavaScript SDK and Graph API services. Once you have implemented this feature, a user can instantly be authenticated onto your Web site using their Facebook credentials, provided that they indicate to Facebook that they trust your Web site.

Your Perl CGI template needs to load the Facebook Connect JavaScript SDK, which provides the Facebook Connect Login button. When clicked, the button automatically opens a dialog box that connects to Facebook, prompts the user for their credentials, and sets an access token. Because Facebook Connect exists only as JavaScript code, your Perl script needs to be able to retrieve details about the connected user. The access token, available to your Perl script in the form of a

cookie, can be used to query the Facebook Graph API service to retrieve information about the connected user.

The Graph API is the latest Facebook security protocol. Linking it into your Perl CGI code is very easy: you use the LWP::Simple Perl module to query it over HTTPS, and the JSON module to decode its output into a Perl hash reference. Both LWP::Simple and JSON should be pre-installed on most Perl distributions. If they are not, you can install either of them using CPAN, or a platform-specific method as described in Chapter 9.

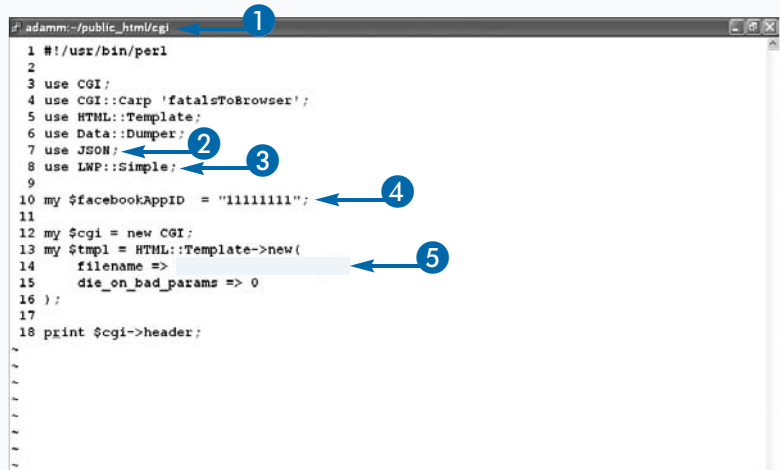
Unfortunately, because Perl is not an officially supported language, this module only supports the older REST API and currently lacks Graph API support. However, manually querying the Graph API is very easy with the help of LWP::Simple and JSON. Additional information is available at <http://developers.facebook.com/docs/authentication/>.

Enable Facebook Connect on Your Web Site

- 1 Open a Perl script in a text editor that uses CGI, HTML::Template, and Data::Dumper.
- 2 Type **use JSON;**
- 3 Type **use LWP::Simple;**

Note: LWP::Simple and JSON do not need a special initialization routine.

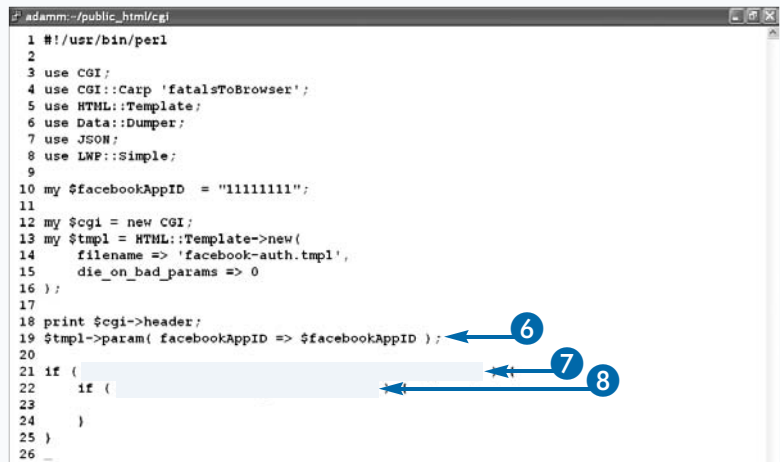
- 4 Type **my \$facebookAppID = "APP_ID";**
- 5 Use the template file `facebook-auth.tmpl`.



```
#!/usr/bin/perl
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use HTML::Template;
6 use Data::Dumper;
7 use JSON;
8 use LWP::Simple;
9
10 my $facebookAppID = "11111111";
11
12 my $cgi = new CGI;
13 my $tmpl = HTML::Template->new(
14     filename =>
15     die_on_bad_params => 0
16 );
17
18 print $cgi->header;
```

- 6 Send your `$facebookAppID` to your template; the JavaScript SDK will use it.
- 7 Retrieve the `fbs_$facebookAppID` cookie.
- 8 Test for the access token value.

Note: Matching regular expression patterns that use brackets will automatically store the matching string in a special variable: `$1`.



```
19 $tmpl->param( facebookAppID => $facebookAppID );
20
21 if (
22     if (
23
24
25 )
26
```

- 9 Type `my $graphURL = "https://graph.facebook.com";`
- 10 Type `my $json = get("$graphURL/me?access_token=$1");`

Note: The `LWP::Simple` module provides `get`, which is a very easy way to make HTTP or HTTPS queries and forward the results into a Perl scalar.

- 11 Type `my $userData = decode_json($json);`

Note: Raw JSON data is very close to a Perl hash in structure, except it does not use any Perl syntax. The function `decode_json` makes its data accessible as a hash reference.

- 12 Type `$tmpl->param($userData);`
- 13 Type `$tmpl->param(debug => Dumper($userData));`

Note: Dumping `$userData` is for debugging purposes only. This shows you everything being returned by the Graph API query.

- 14 Type `print $tmpl->output;`
- 15 Save the Perl CGI script.

```
#!/usr/bin/perl
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use HTML::Template;
6 use Data::Dumper;
7 use JSON;
8 use LWP::Simple;
9
10 my $facebookAppID = "11111111";
11 my $graphURL = "https://graph.facebook.com";
12
13 my $cgi = new CGI;
14 my $tmpl = HTML::Template->new(
15     filename => 'facebook-auth.tmpl',
16     die_on_bad_params => 0
17 );
18
19 print $cgi->header;
20 $tmpl->param( facebookAppID => $facebookAppID );
21
22 if ( my $token = $cgi->cookie( "fbs_$facebookAppID" ) ) {
23     if ( $token =~ /access_token=(.*)$/ ) {
24         my $json = get( "$graphURL/me?access_token=$1" );
25         my $userData = decode_json( $json );
26         $tmpl->param( $userData );
27     }
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
#!/usr/bin/perl
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use HTML::Template;
6 use Data::Dumper;
7 use JSON;
8 use LWP::Simple;
9
10 my $facebookAppID = "11111111";
11 my $graphURL = "https://graph.facebook.com";
12
13 my $cgi = new CGI;
14 my $tmpl = HTML::Template->new(
15     filename => 'facebook-auth.tmpl',
16     die_on_bad_params => 0
17 );
18
19 print $cgi->header;
20 $tmpl->param( facebookAppID => $facebookAppID );
21
22 if ( my $token = $cgi->cookie( "fbs_$facebookAppID" ) ) {
23     if ( $token =~ /access_token=(.*)$/ ) {
24         my $json = get( "$graphURL/me?access_token=$1" );
25         my $userData = decode_json( $json );
26         $tmpl->param( $userData );
27         $tmpl->param( debug => Dumper( $userData ) );
28     }
29 }
30
31 print $tmpl->output;
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Apply It

The Facebook Graph API simplifies the way you query and submit Facebook data. In this section's example, you are only using it to query the `me` object, which will always refer to the user who is currently logged in. You can learn more about interacting with the Graph API from the Facebook Developers' documentation page at <http://developers.facebook.com/docs/api>.

TYPE THIS

```
$cgi->cookie( "fbs_$facebookAppID" ) =~ /
access_token=(.*)&?/;
my $token = $1;
sub QueryGraphAPI {
    my ( $path, $params ) = @_;
    my $json = get( "$graphURL/$path?access_
token=$token&$params" );
    return decode_json( $json );
}
my $groups = QueryGraphAPI( "me/groups" );
```

RESULTS

You can query arbitrary objects using this `QueryGraph` subroutine. In this case, the object `me/groups`, according to the Graph API documentation, retrieves a list of all groups of which the currently logged-in user is a member.

Enable Facebook Connect on Your Web Site (continued)

The JSON data output produced by the Graph API query is completely compatible with HTML::Template, once you convert it into a Perl hash reference. This makes it really convenient to display the JSON array because it automatically converts itself into a format compatible with `TMPL_LOOP`. However, if you make multiple queries to the Graph API, there is a chance that the same array key could be used multiple times. This makes it impossible to chain two JSON outputs into a single template without changing one of the `TMPL_LOOP` key names.

In this section, you will load the JavaScript SDK in a slightly different way from earlier in this chapter. You want to load the SDK synchronously so the user's Web browser waits for it to be loaded before displaying your

Web site. In the SDK, you also subscribe to the `auth.login` event, and tie it to a page reload command. This ensures that your Web page is reloaded after the user has successfully authenticated to Facebook, and indicates that they trust your Web site.

The Facebook Connect Login button is provided by an XFBML tag inside your HTML::Template file. Because the JavaScript SDK renders XFBML, the only thing that matters is its placement in your Web site template:

```
<fb:login-button></fb:login-button>
```

You can apply additional attributes to the button, which allow you to control its text, display background, and other cosmetic options. You can access the button's documentation at <http://wiki.developers.facebook.com/index.php/Fb:login-button>.

Enable Facebook Connect on Your Web Site (continued)

16 Open the HTML template referenced by your Perl CGI script.

17 Type `<fb:login-button autologoutlink="true"></fb:login-button>`.

"This is the JavaScript SDK FBML. It will be converted into the Facebook Connect Login button.

18 Type `<tmpl_if name=id> </tmpl_if>`.

Note: This `id` template value will resolve to the user's Facebook Profile ID number, but only when the user is logged in using Facebook Connect.

19 Type `<pre><tmpl_var name=debug></pre>` to display Data::Dumper JSON output.

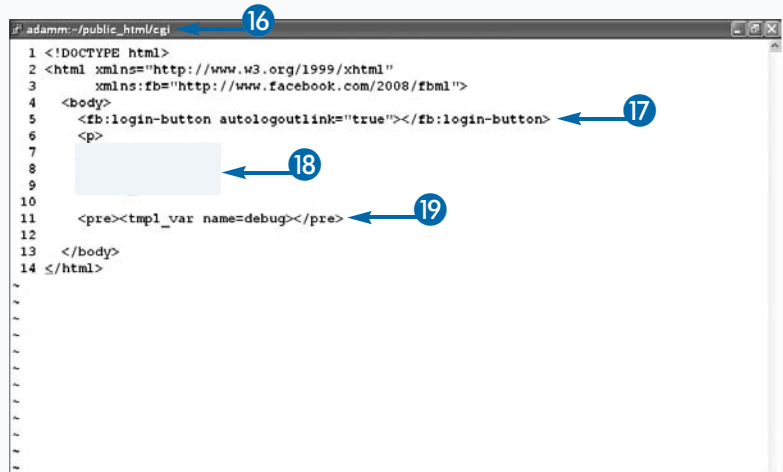
20 Use data provided by JSON to personalize the logged-in user's message.

21 Import Facebook Connect's JavaScript SDK code synchronously.

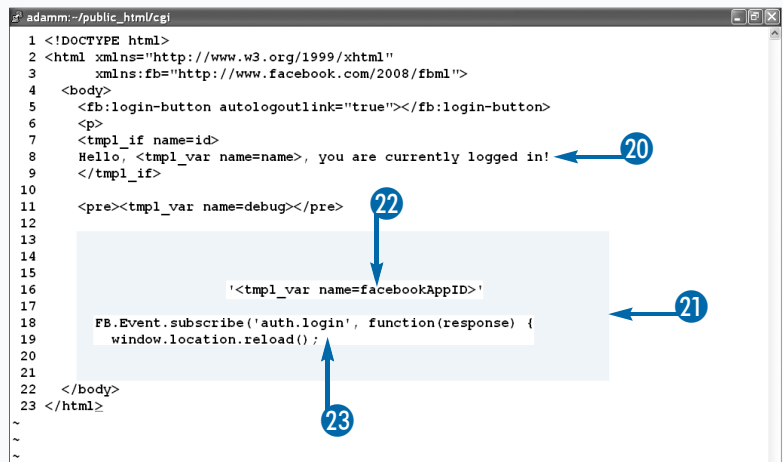
22 Type `<tmpl_var name=facebookAppID>` into the SDK's initialization function.

23 Subscribe to the `'auth.login'` event.

24 Save your template.

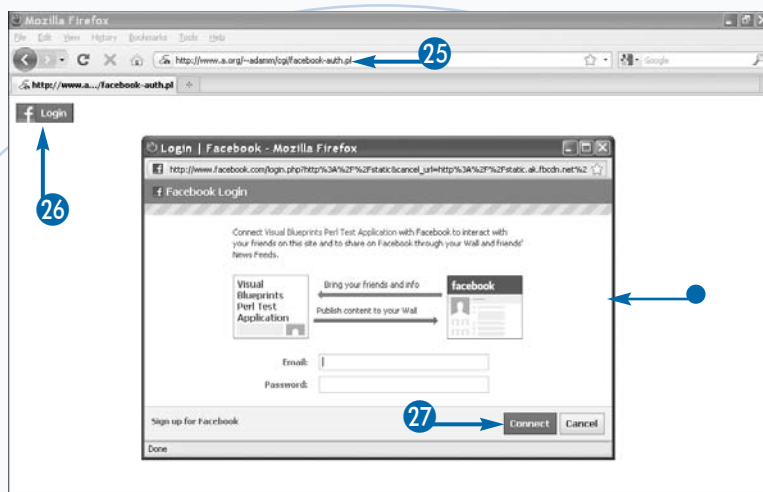


```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:fb="http://www.facebook.com/2008/fbml">
4   <body>
5     <fb:login-button autologoutlink="true"></fb:login-button>
6     <p>
7       <pre><tmpl_var name=debug></pre>
8   </body>
9 </html>
```



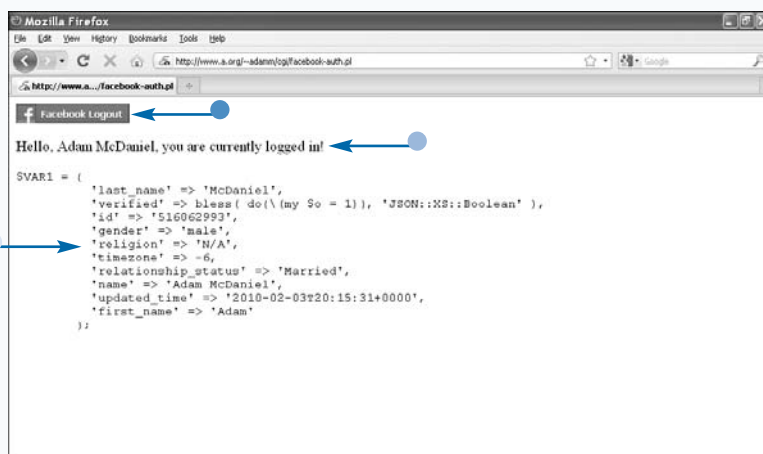
```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:fb="http://www.facebook.com/2008/fbml">
4   <body>
5     <fb:login-button autologoutlink="true"></fb:login-button>
6     <p>
7       <tmpl_if name=id>
8         Hello, <tmpl_var name=name>, you are currently logged in!
9       </tmpl_if>
10     <pre><tmpl_var name=debug></pre>
11     <pre><tmpl_var name=facebookAppID>
12     FB.Event.subscribe('auth.login', function(response) {
13       window.location.reload();
14     });
15   </body>
16 </html>
```


- 25 Open your Perl CGI script in a browser.
- 26 Click the Facebook Connect Login button.
 - The Facebook Authentication dialog box appears.
- 27 Type in your Facebook e-mail address and password and click Connect.



Your Perl CGI automatically reloads.

- The Login button changes to a Logout button.
- The browser displays the custom login message.
- The browser displays the debug JSON output.



Extra

If you are already logged into your Facebook profile, you can skip steps 26 and 27. Likewise, if you click the Facebook Logout button now, you are also logged out of your Facebook profile. This is the nature of single sign-on: your actions on one Facebook Connect site directly affect your state on other linked sites.

If you are only interested in using Facebook Connect for authentication, you can safely query the Graph API's `me` object, and only use the `id` value as the user's unique identifier. Your CGI code can trust that the `id` value is unique, and cannot be impersonated, as long as you trust Facebook Connect.

The amount of information that is actually available on the Graph API is immense. It is an alternate gateway to all Facebook data. This actually makes using Facebook on third-party applications and Web sites very scary: any information you have submitted into your Facebook profile is now accessible, and without your immediate consent.

Because Facebook is constantly attacked by privacy groups, and rightly so, expect this API to change and evolve over time. The information and techniques listed in this chapter are accurate as of May 2010. You can find more information about the current Facebook privacy policy at <http://developers.facebook.com/policy/>.

Understanding the Facebook Canvas Feature for Applications

The Facebook Canvas feature allows you to create a Facebook application that users of the popular social media Web site can access and share with their friends. An application is basically a CGI program hosted on your Web servers but controlled by Facebook. Once you create the program using the Canvas interface, users can access it through a specially wrapped URL:
`http://apps.facebook.com/myapp/`.

Documentation for the Facebook Canvas feature is available at `http://developers.facebook.com/docs/`

`guides/canvas/`. Note that this guide assumes that PHP is your development language. The instructions and theory are still relevant to Perl, as long as you ignore the PHP-specific code. The code can be provided by a third-party Perl module available on CPAN, called `WWW::Facebook::API`. This module allows you to use Perl to develop your Canvas page. The module includes examples for implementing it, as well as additional documentation specific to Perl.

Preparation

Before you begin, you need to configure your Facebook Application as a canvas. Go to the Facebook Developers' applications page at `www.facebook.com/developers/apps.php`, click Edit Settings, and then click Canvas. Once you have set the Canvas Page URL, Canvas Callback URL, and Render Method, you can begin developing your application's Perl CGI script.

Canvas Page URL

The Canvas Page URL represents your unique path in the Facebook Apps subdomain. This value must be one word, lowercase, and only contain letters, numbers, dashes, and underscores. This URL is your Application Path, which Perl will use later.

Canvas Callback URL

The Canvas Callback URL is the path to your application CGI script hosted on your Web server. Your CGI script must be housed on a valid hosting provider. For more information on finding a hosting provider, see Chapter 1. Note that the Canvas Callback URL must end in a forward slash. This may look unusual but it is intentional:

```
http://www.mydomain.com/cgi-bin/my-facebook-app.pl/
```

Render Method

Facebook allows for two render methods: `iframe` and `FBML`. The `iframe` is an HTML tag that superimposes one Web site into another. This is the older way of displaying legacy Web sites that were not designed for Facebook within the Facebook interface. The `FBML`, or Facebook Markup Language, option is a set of custom tags developed by Facebook to make it easier for third-party applications to use Facebook features. If you are just starting out, you should use `FBML`.

Using Perl

You can use Perl to connect to the Facebook Canvas API with the `WWW::Facebook::API` module. You can find details about the module at `http://wiki.developers.facebook.com/index.php/User:Perl`. After you install the module, several pages of documentation become available through `Perldoc`. You can find a complete list of submodules and their descriptions in Appendix D.

Initialization

When first loading the `WWW::Facebook::API`, you need to use the `new` constructor to initialize the module. The constructor allows for several attributes when you initialize it. You can set these attributes as options to `new`, or as separate methods later. At a minimum, you should specify the following attributes as options:

```
my $facebook = WWW::Facebook::API->new(
    api_key => 'Application Key',
    secret => 'Application Secret',
    app_path => 'Application Path',
);
```

The `new` constructor accepts other attributes, which you can also set as methods. For example, if you want to enable debugging midway through your program, you can use something like this:

```
$facebook->debug( 1 );
```

You can find a complete list of available attributes in the module's PerlDoc page; scroll down to Attribute Methods for a list and description.

Authentication and Validation

Your Facebook Application CGI needs to have the correct API key and secret values in order to communicate to Facebook. This happens automatically when you initialize the module, specify the correct `api_key` and `secret` attributes, and then call the `validate_sig` method:

```
my $cgi = new CGI;
my $params = $facebook->canvas->validate_sig(
    $cgi );
```

Because the validation process involves decrypting the fields submitted by Facebook with your App Secret value, `validate_sig` requires access to the CGI module's reference scalar. The output of this process is the decrypted parameters that are relevant to your session.

API Calls

Once a user has trusted your application, which they typically do by adding it to their Facebook profile, you are allowed to make specific API calls to the Facebook servers, depending on what they trust you with.

Facebook Interaction

Using `WWW::Facebook::API`, you can interact with nearly the entire published Facebook API.

The Graph API

The Graph API is the newest Facebook programming interface, designed to simplify communication to the Facebook servers. Because `WWW::Facebook::API` actually uses the older REST API, all the new features available in the Graph API are not directly available from the Perl module. Fortunately, the Graph API is very easy to connect to. You only need a way to query an HTTPS source, and to decode the JSON output. The `LWP::Simple` and `JSON` Perl modules are a natural fit for connecting to the Graph API. You can access online documentation for the Graph API at <http://developers.facebook.com/docs/api>.

Facebook Markup Language

FBML, or Facebook Markup Language, is a customized series of XHTML-like tags that allow for easy access to Facebook functionality on their application. Two implementations of FBML are available for Facebook applications: a server-side JSON version and a client-side JavaScript SDK version. If you have already implemented any Facebook Social plugins into your Web site, you are already configured to use the JavaScript method. Keep in mind that if you chose to implement the JavaScript SDK version of FBML, you need to configure your Facebook Application settings to use the `iframe` rendering method, not FBML.

You can access online documentation for the server-side JSON version of FBML at <http://developers.facebook.com/docs/reference/fbml/>. You can find documentation for the client-side JavaScript SDK version of FBML at <http://developers.facebook.com/docs/reference/javascript/>.

Facebook Query Language

FQL, or Facebook Query Language, is an SQL-like language that Facebook developed to allow for easy access to the Facebook back-end database. FQL is only available on the server-side JSON interface, which you can manage using the `WWW::Facebook::API::FQL` module.

The syntax of the query language is very close to SQL, but it is not true SQL as you would find in a MySQL or Oracle database. The following code is the basic structure of FQL, where the `fields`, `table`, and `conditions` values can be customized:

```
select fields from table where conditions
```

Some of the tables you can query from include `comment`, `event`, `friend`, `group`, `message`, `photo`, `status`, `user`, and `video`. This is all assuming that the user has granted your application the rights to do so.

Create a Facebook Application with Perl

You can create a Facebook Application with a Perl CGI script using the `WWW::Facebook::API` module and the Facebook Canvas interface. Users can add your application to their profile, and then access your CGI code from a Facebook Apps URL, or directly from a bookmark on their Facebook profile home page.

Before you can proceed, your CGI must be registered as a Facebook Application, and correctly configured with a Canvas Page URL, Canvas Callback URL, and specific settings relevant to `WWW::Facebook::API`. The Canvas Page URL is the public URL that users see under the Facebook Apps subdomain, `http://apps.facebook.com/myapplication/`. You can give this URL directly to people if you want. It automatically logs a user onto Facebook before displaying your application.

Create a Facebook Application with Perl

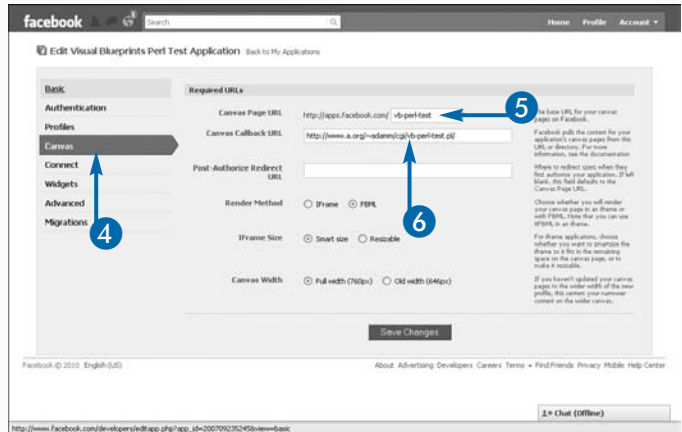
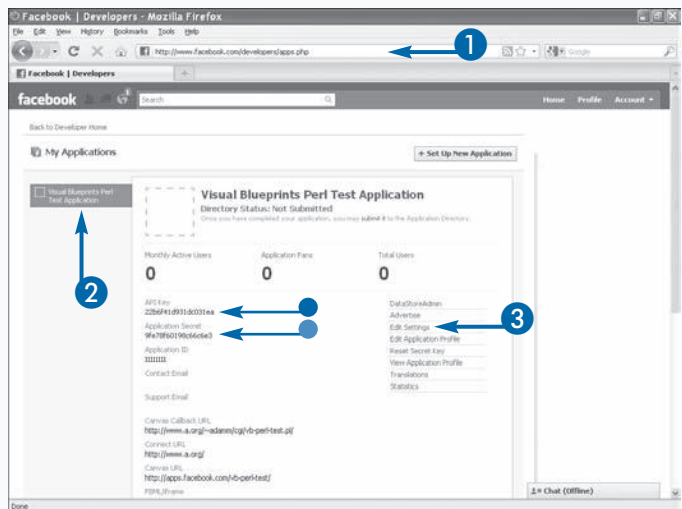
- 1 Go to the URL `www.facebook.com/developers/apps.php`.
- 2 Select your App.
 - Your API Key value.
 - Your Application Secret value.
- 3 Click Edit Settings.

- 4 Click Canvas.
- 5 Type in your Canvas Page URL.
- 6 Type in your Canvas Callback URL.

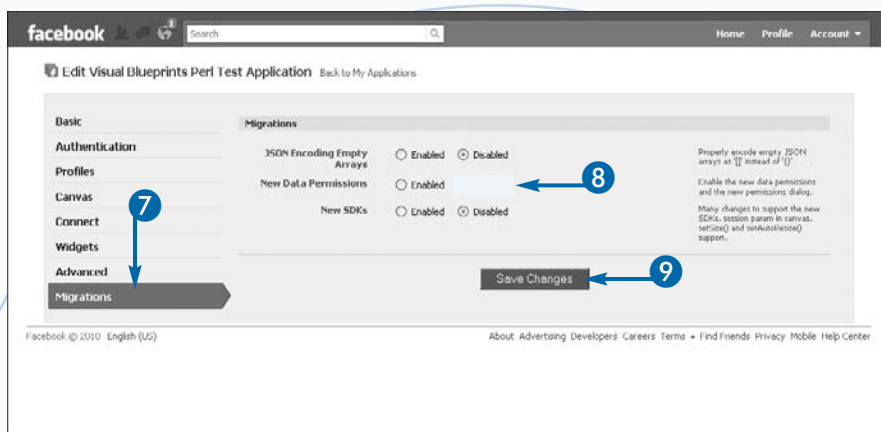
Note: This address must end in a forward-slash.

The Canvas Callback URL determines where Facebook can access your CGI code. This must be hosted at a public Web server that Facebook can access. The format of this URL has one unique quirk: it must end in a forward-slash. `WWW::Facebook::API` has examples that demonstrate how URLs with content after the forward-slash reference subpages on your application CGI. You also need to install the `WWW::Facebook::API` module from CPAN. For more information, see Chapter 9.

Unfortunately, because Perl is not an officially supported development language for Facebook, it lags a little behind the latest API features that are available on Facebook. For this reason, you need to instruct Facebook to disable the New Data Permissions. Eventually, `WWW::Facebook::API` will be updated to the new permissions properly, but for now you need to disable these permissions in order to make your Perl Application work.

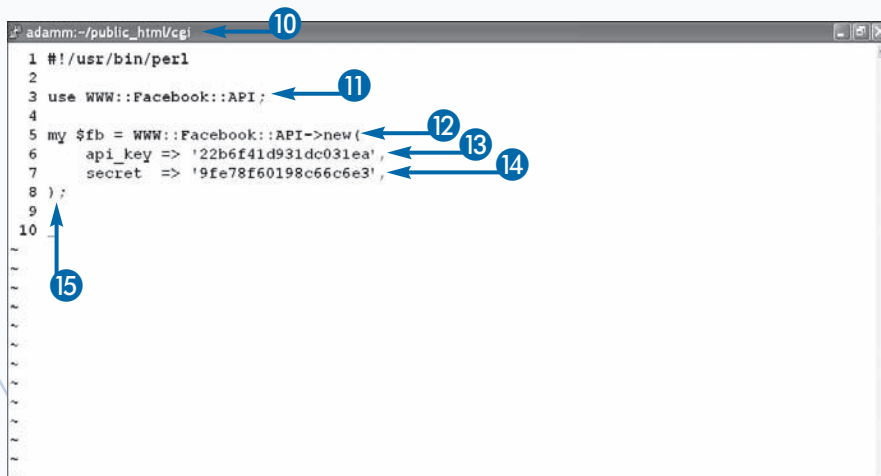


- 7 Click Migrations.
- 8 Click to select the Disabled option for New Data Permissions.
- 9 Click Save Changes.



- 10 Open a Perl CGI script.
- 11 Type `use WWW::Facebook::API;`
- 12 Type `my $fb = WWW::Facebook::API->new(`
- 13 Type `api_key => 'API Key',`
- 14 Type `secret => 'Application Secret',`
- 15 Type `);`

Note: From here, you can use `$fb` to access all of `WWW::Facebook::API`'s methods.



Extra

An example Perl script that uses the Facebook Canvas API is available from the authors of `WWW::Facebook::API`. You can download it by typing <http://cpansearch.perl.org/src/UNOBE/> in your browser and pressing Enter. Click the latest `WWW-Facebook-API` version, click examples, and then click `facebook-canvas-json`.

If you download this Perl CGI example, be sure to configure your correct API Key and Application Secret values; otherwise, Facebook will not be able to communicate to it over the Canvas interface.

It is beyond the scope of this book to describe for you exactly how to create a Facebook Application, feature by feature. There are several Web sites, written by third-party and Facebook developers, that describe in detail everything that you need to know. The trick to understanding this documentation is to identify the `WWW::Facebook::API` equivalent whenever any PHP or Java code is referenced.

Introducing the Twitter APIs

Twitter has several application programming interfaces (APIs) that you can use to interact with users and feeds of the social media platform. If you were to compare the features available through the Twitter Web site against their APIs, you would find there are more Twitter features available through third-party apps than are offered by the Twitter Web site.

The protocol used to communicate to these APIs is called REST, or Representational State Transfer. Using the HTTP protocol, Twitter provides a specific URL for each API service, where several REST requests can be made in parallel to generate the complete picture of a user's

current state on Twitter. As a response, REST outputs XML or JSON back as your request results. You need to use Perl to examine this data to determine whether or not your request was successful. Within each REST session, you need to authenticate your application to Twitter. You can do this using the legacy basic authorization method, or the more secure OAuth method.

Fortunately, there are some Twitter-specific Perl modules that automate the REST communication and authentication aspects of the available APIs. You can find the complete API documentation online at <http://dev.twitter.com/doc/>.

Authorization Protocols

Twitter needs to know who you are when you use its APIs, and which Twitter user profile you are representing. You must use some form of authorization when communicating over REST that introduces you correctly.

Basic Authorization

Basic authorization involves sending a username and password for every API request to Twitter's servers. This method is weak because it requires that users trust their credentials to a third-party to keep secure.

As a result, basic authentication was disabled on the Twitter API in July 2010. Developers are expected to use the OAuth protocol within REST to manage authentication and communication.

OAuth

OAuth is a protocol designed for secure authorization and communication because all API requests are signed by a special access token. This access token is only granted after you create a request token and redirect the user for authorization directly on the Twitter Web site. To manage both of these tokens, you are assigned private and public keys that are used for signing and authenticating traffic between your Web site and Twitter. After you register your Perl application on the Twitter development platform Web site, you are assigned OAuth-specific keys that you can plug into Perl. Twitter recommends OAuth for all third-party client authentications because it eliminates the need for third parties to manage end-user credentials directly.

Twitter API Services

Twitter supports various APIs, each using different protocols and authentication methods. Listed here are two that are based on REST, one that uses a persistently connected HTTP socket, and one based on JavaScript.

Twitter API

You can use the Twitter API to communicate on behalf of a user to access their Twitter profile using your interface instead of the standard Twitter Web site. The Twitter API uses the REST protocol for communication, but also requires some form of authorization, using either the legacy basic authorization or, preferably, the newer OAuth method. This ensures that Twitter recognizes your app, and that you have authorization to represent a specific Twitter user.

You can use the Twitter API to query various resources linked to Twitter profiles, including users' timelines, tweets, trends, lists, messages, friends, favorites, and notification settings. This chapter demonstrates how to use the Twitter API within Perl in these contexts and applications. You can learn more about how to use the Twitter API at <http://dev.twitter.com/pages/intro-to-twitterapi/> and http://dev.twitter.com/pages/api_overview/.

Twitter API Services (continued)

Search API

The Search API allows you to search past tweets on the Twitter network, so that you can identify trending topics, query specific hash-tags, and search for specific terms that other people have tweeted about.

The Search API also uses the REST protocol, but it is kept separate from the original Twitter API because it does not require any form of authentication. In other words, when searching for specific Twitter activity, you can do it anonymously. For more information on how to use the Search API, go to <http://dev.twitter.com/doc/get/search/>.

Streaming API

The Streaming API offers near real-time access to Twitter statuses, delivered through a persistently connected HTTP request. This means that your application can listen for status updates as they are posted to Twitter, through an HTTP connection that rarely terminates.

According to the Streaming API documentation, Twitter keeps the connection open for as long as is practical, barring any server-side errors, client-side lags, network problems, or server maintenance.

There is no Perl-specific module that links into the Streaming API, but it is not difficult to develop an application that keeps HTTP open and active using Net::HTTP (a low-level HTTP development module for Perl). For more information on how the Streaming API works and its caveats, go to http://dev.twitter.com/pages/streaming_api/. An example program to connect to the Streaming API with Net::HTTP is described later in this chapter.

@Anywhere

@Anywhere is the Twitter JavaScript API designed to easily add Twitter functionality to external Web sites. By importing the JavaScript code into any of your HTML or CGI pages, you can easily add some interesting Twitter features with little effort. You can find more information about @Anywhere at <http://dev.twitter.com/anywhere/begin>.

Auto-Link User Profile

The auto-linking feature monitors your Web site's content for Twitter usernames, and automatically converts them into a link to that user's Twitter profile. This way, you only need to reference @username in your Web site, and JavaScript will automatically expand it into `@username`.

Hovercard

A Hovercard extends the auto-link feature. It adds a dynamic popup window that appears when a user hovers their mouse over the link. The user's profile picture appears in the Hovercard, along with their biography, latest tweet, and a Follow Me button.

Follow Me Button

A Follow Me button is a link on your Web site that allows other users to follow your Twitter profile from theirs, with a single mouse-click. This makes it convenient for your users to follow you without actually going to `Twitter.com`, searching for your Twitter username, and clicking the follow-me link there.

TweetBox

A TweetBox is like a self-contained Twitter client for your Web site, written entirely in JavaScript. Adding this feature allows your users to post tweets to their own account from your Web site. You can use this feature to create pre-composed tweets that promote a specific topic or event. The user has the option of customizing the message before clicking the Tweet button to make the text appear on their personal Twitter feed.

Introducing the Perl Twitter Modules

Unlike Facebook, the support for Twitter in Perl is much more robust and mature. There are actually several modules you can use to make creating a Twitter-friendly application easier for you to develop for your Web site. You can install all of these modules using CPAN, or, if available, using your platform's packaging system. You can find instructions on how to do this in Chapter 9.

Each module has different interface levels, so choosing the right one depends on your application. If you are looking for a way to query friends and timelines, `Net::Twitter::Lite` is your best option. If you just want to post a tweet, try `App::Tweet`. Or, if you want something

like Facebook Connect to handle authentication, using `Net::OAuth` directly is also an option.

Deciding on which module to use, and how to use it, will be a matter of experimentation. Like Facebook, Twitter does not officially support Perl as a development API platform, but other Perl developers have made a decent effort to provide an interface for Twitter using these modules.

Before you can begin coding anything, you need to have registered your Web site as an application on the Twitter development platform. This assigns for you a series of keys that you can use within any Perl scripts you develop.

Perl Twitter Modules

A few other Perl developers have produced “front-end” modules for the Twitter API that manage the communication protocols automatically.

Net::Twitter

The `Net::Twitter` module is the primary Perl module used to access various Twitter APIs. This module features a configuration mechanism that is not very common in Perl, called a *trait*. When you initialize the module, you can define which traits should be activated, such as `OAuth` for OAuth authentication, `API::REST` to use the Twitter API standard, and `API::Search` to use the Search API standard. Note that this module does not currently support the Streaming API.

You also need to provide your consumer key and consumer secret during initialization. These values are assigned to you after you register your Web site as a Twitter application. Twitter uses them to keep track of where their public APIs are being deployed, and by whom:

```
use Net::Twitter;
my $tw = Net::Twitter->new(
    traits => [ qw( TRAITS ) ],
    consumer_key => KEY,
    consumer_secret => SECRET,
);
```

`Net::Twitter` supports nearly every Twitter API and Search API function, and even adds support for a third-party API called *Twittervision*. Produced outside of Twitter, *Twittervision* is a combination of tweets and Google Maps. You can find additional information at www.twittervision.com/api.html.

```
eval{
    VAR = $tw->function( { ARGS } );
    # Examine VAR for returned function results.
}
if ( $@ ) {
    # Process errors with Net::Twitter::Error
}
```

`Net::Twitter` uses a different mechanism for sending error messages if there is a problem: it throws exception signals. In case an API call encounters an error, the module actually kills off itself and the running program. In doing so, it expects developers to wrap their API functions in an `eval` block, which traps the exception signal, allowing the module to die while leaving your program running. After the module dies,

Perl Twitter Modules *(continued)*

you can access the actual error code and error message through a special variable: `$@`. If you look at the submodule `Net::Twitter::Error` and its PerlDoc page, you can find more information about this particular technique.

If you do not like this exception process, you can apply the `WrapError` trait when initializing the module. This causes all `Net::Twitter` methods to return `undef` on failure, rather than kill itself and your program; you then access the error code through the module's `get_error` method.

You can find out more information about `Net::Twitter` from its CPAN page at <http://search.cpan.org/dist/Net-Twitter/>.

Net::Twitter::Lite

As the name implies, `Net::Twitter::Lite` is a lightweight version of `Net::Twitter`. If the larger module offers too many features for your application, this scaled-down version may be a better option.

`Net::Twitter::Lite` uses `Net::Twitter` with the `API : REST` and `OAuth` traits enabled. This means that you can use the Twitter API standard with either OAuth or basic authentication.

App::Tweet

`App::Tweet` is a simple wrapper for the `Net::Twitter` module, providing the fastest way to produce a command-line Twitter client in Perl. It features a built-in configuration script that prompts for credentials automatically the first time it is run:

```
use App::Tweet;
App::Tweet->run( message => message );
```

Unfortunately, this configuration script only interfaces with a terminal and so you cannot use it with Apache as a CGI program. As a result, it is only suitable for a command-line Twitter application.

Once configured, you can easily post tweets using your own Twitter account with the `run` method. There is even a `reconfigure` method that allows you to update your saved credentials for subsequent program runs.

Register a New Twitter Application

You can register your Web site as a Twitter application to access the Twitter API and @Anywhere JavaScript API. Registration also introduces your Web site to the Twitter community. To begin the registration process, go to <http://dev.twitter.com/apps/new/>.

The application registration form asks you for an application name, description, and whether your application is a client-side program or a browser-based CGI. The decision of client or browser affects how OAuth handles user authentications later. If you register as a client application type, a browser will be created during the authentication handover process. If you register as a browser, Twitter will redirect the user within their current browser window, then back to your Web site, using the configured Callback URL.

Register a New Twitter Application

- 1 Open a Web browser to <http://dev.twitter.com/apps/new>.

The Twitter Register an Application form appears.

- 2 Enter your application's details.
- 3 Select the Browser application type.

Note: If you want to compare the two access types, you can register a second application as a client application type.

- 4 Enter your Callback URL.
- 5 Select the Read & Write access type.
- 6 Type in the Captcha.
- 7 Click Register application.

Once you are registered, your browser displays an application summary screen. Here you will find some keys, secrets, and URLs that are specific to your application.

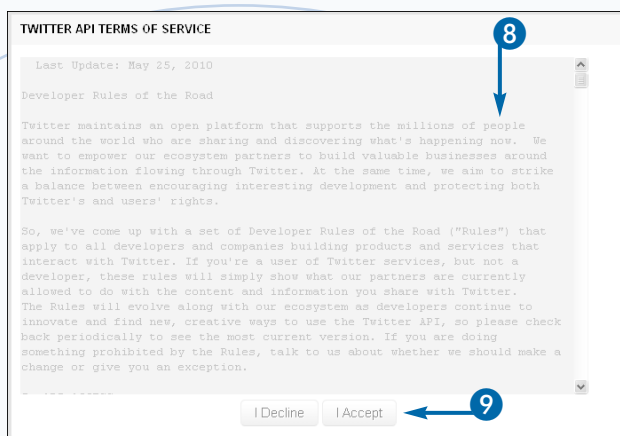
The API key is used by the @Anywhere JavaScript API. If you are only interested in @Anywhere and do not plan to develop a Net::Twitter Perl CGI program, you can skip to the section, "Use the Twitter @Anywhere JavaScript API."

The *consumer key* is a public key that is unique to your application. The *consumer secret* is like a passcode that is specific to your consumer key. Later, you will use both within Net::Twitter to gain access to Twitter through OAuth authorization, and to submit requests on behalf of a user who has logged into your CGI program.

At any time, you can review your registered apps and current configuration settings at <http://dev.twitter.com/apps/>.

The Twitter API Terms of Service document appears.

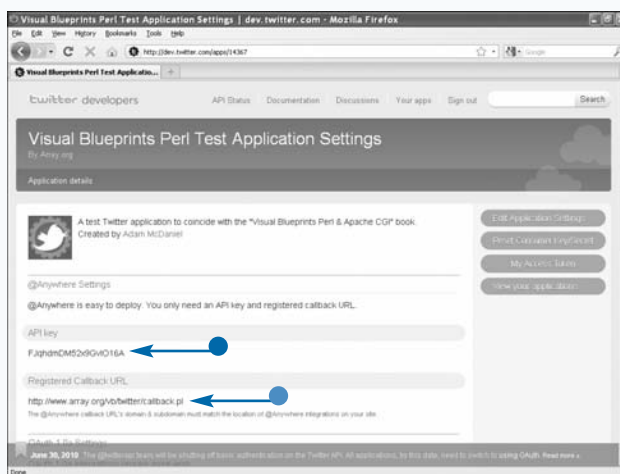
- 8 Read the document.
- 9 Click I Accept.



The application summary screen appears.

- Your API key.
- Your Registered Callback URL.

You can scroll down for the OAuth Consumer key, secret, and other OAuth URLs.



Extra

The registration process provides you with several OAuth keys, tokens, secrets, and URLs. If you plan to use Net::Twitter, you only need to worry about the consumer key and consumer secret values; they are already preprogrammed into the module. You may want to reference the actual Twitter URLs if you plan on using Net::OAuth directly.

The Callback URL will become a Perl CGI script that you will develop later in this chapter. If you do not know its correct URL value now, you can always come back to this screen later and update it.

On the right side of the application summary screen you will find a link called My Access Token. This is a pre-calculated personal access token and access token secret that is already linked to your personal Twitter profile. If you plan on creating a private Twitter application that only you will use, you can bypass the entire OAuth authentication process and gain immediate access to your own account.

Do not divulge your access token and access token secret values. Otherwise, anyone can impersonate you on Twitter.

Authenticate to Twitter Using OAuth

The OAuth authentication process is not overly complicated, but the entire process must proceed exactly as designed; otherwise, your Perl CGI scripts will not be granted access to the Twitter API. Note that OAuth is easier to work with if you use the `Net::Twitter` Perl module, as opposed to implementing `Net::OAuth` directly. Your Perl CGI script must first import the `Net::Twitter` module, and initialize it with the consumer key and consumer secret values you received during registration. There are three stages of OAuth authentication, so your Perl CGI will need to identify what stage the user is currently at and process them to the next stage, accordingly. When a user first visits your Web page, the user default to the first stage of OAuth: *not authorized*. You must generate a request token and request token secret, combine these

values with your consumer key, and redirect the user to the Twitter OAuth URL. Here, the user types in their credentials and clicks the Allow button, authorizing their account to be accessed from your Web application. Twitter then redirects the user's browser to your Callback URL.

The second stage of OAuth, *request access*, begins when you first respond to the Callback URL request. You receive from Twitter two new CGI parameters validating the preceding step. Called `oauth_token` and `oauth_verifier`, both parameters must be combined with the original request token secret and sent back to the Twitter OAuth server. If Twitter is satisfied, you receive an access token and access token secret.

The third and final stage of OAuth is *access granted*. The access token and access token secret values are used to access the Twitter API functions on behalf of the user.

Authenticate to Twitter Using OAuth

- 1 Type `use Net::Twitter;`
 - 2 Type `my $tw = Net::Twitter->new(`
 - 3 Type `traits => [qw(API::REST OAuth)],`
 - 4 Type `consumer_key => KEY,`
 - 5 Type `consumer_secret => SECRET,`
 - 6 Type `);`
 - 7 Create an `authorize_user` subroutine.
This will handle your first stage of the authentication process.
 - 8 Type `my $url = $tw->get_authorization_url(callback => URL);`
 - 9 Store `$tw->request_token` and `$tw->request_token_secret` into a session cookie.
- Note:** Make sure you have imported the CGI module and initialized it as `$cgi`.
- 10 Use CGI's `redirect` method to redirect the user to `$url` and assign the cookie.

```
adam@array3:~/public_html/vb/twitter
1 #!/usr/bin/perl
2
3 use strict;
4 use Net::Twitter;
5 use CGI;
6 use CGI::Carp qw(fatalsToBrowser);
7 use Data::Dumper;
8
9 my $cgi = new CGI;
10 my $tw = Net::Twitter->new(
11     traits => [qw(API::REST OAuth)],
12     consumer_key => '1111222233334444',
13     consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
14 );
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

```
adam@array3:~/public_html/vb/twitter
1 #!/usr/bin/perl
2
3 use strict;
4 use Net::Twitter;
5 use CGI;
6 use CGI::Carp qw(fatalsToBrowser);
7 use Data::Dumper;
8
9 my $cgi = new CGI;
10 my $tw = Net::Twitter->new(
11     traits => [qw(API::REST OAuth)],
12     consumer_key => '1111222233334444',
13     consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
14 );
15
16
17 sub authorize_user
18
19
20
21
22 warn "Getting Authorization URL: " . $url;
23 warn "Getting Request Token: " . $tw->request_token;
24 warn "Getting Request Token Secret: " . $tw->request_token_secret;
25
26
27
28
29
30
31 print $cgi->redirect( -uri => $url, -cookie => $cookie );
32 }
```


- 11 Read the session cookies into the %sess hash.
- 12 Look for "oauth_token" and "oauth_verifier" CGI parameters, and store them as \$request_token and \$verifier.
- 13 Type `$tw->request_token($request_token);`.
- 14 Type `$tw->request_token_secret($sess{'request_token_secret'});`.
- 15 Type `$tw->request_access_token(verifier => $verifier);`.
- 16 Store `$tw->access_token` and `$tw->access_token_secret` into the session cookie.
- 17 Use CGI's `redirect` method to redirect the user to your Web site's absolute URL and assign the cookie.

Note: This second redirection is optional, but it clears the URL query string from the "oauth_token" and "oauth_verifier" strings that Twitter supplied.

- 18 Type `else { &authorize_user(); }`.

Note: This last `else` block catches users just starting out at stage one.

```
adam@array3:~/public_html/vb/twitter
10 my $tw = Net::Twitter->new(
11     traits => [qw(API::REST OAuth)],
12     consumer_key => '11111222233334444',
13     consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
14 );
15
16 my %sess = $cgi->cookie( 'sess' );
17
18
19
20
21 $tw->request_token( $request_token );
22 $tw->request_token_secret( $sess{'request_token_secret'} );
23
24 $tw->request_access_token( verifier => $verifier );
25
26 warn "Getting Request token: " . $request_token;
27 warn "Getting Request token verifier: " . $verifier;
28 warn "Setting Access token: " . $tw->access_token;
29 warn "Setting Access token secret: " . $tw->access_token_secret;
30
31
32
33 sub AuthorizeUser {
34     my $url = $tw->get_authorization_url(
35         callback => "http://$ENV{'HTTP_HOST'}$ENV{'SCRIPT_NAME'}",
36     );
37
38     warn "Getting Authorization URL: " . $url;
39     warn "Getting Request Token: " . $tw->request_token;
40     warn "Getting Request Token Secret: " . $tw->request_token_secret;
41 }
```

```
adam@array3:~/public_html/vb/twitter
14 );
15
16 my %sess = $cgi->cookie( 'sess' );
17
18 if ( my ( $request_token, $verifier ) = (
19     $cgi->param( "oauth_token" ),
20     $cgi->param( "oauth_verifier" ) ) ) {
21     $tw->request_token( $request_token );
22     $tw->request_token_secret( $sess{'request_token_secret'} );
23
24     $tw->request_access_token( verifier => $verifier );
25
26     warn "Getting Request token: " . $request_token;
27     warn "Getting Request token verifier: " . $verifier;
28     warn "Setting Access token: " . $tw->access_token;
29     warn "Setting Access token secret: " . $tw->access_token_secret;
30
31
32
33
34
35
36
37
38
39 }
40
41
42
43
44
45 sub authorize_user {
```

Extra

You do not need to provide the Callback URL again into your Perl CGI script, even though you provided it when registering your application. In fact, the two values do not even need to match; the version in your Perl script takes priority.

When you do not define a Callback URL, you are instructing Twitter to treat your application like a desktop app. The difference here is that Twitter does not pass control back to your script with `oauth_token` and `oauth_verify` parameters, as it does in step 12. Instead, Twitter gives the user a special PIN, which effectively acts as the `$verifier` value. Twitter expects you to prompt the user for this PIN, and supply it into step 15.

Throughout this process, cookies are a convenient way to link users to the various tokens and secrets as they progress through each OAuth stage. As these values are represent private security information, storing them as a cookie is not a great long-term idea. Instead, store this information into a local database, and then assign a single, user-specific, uniquely identifying cookie as a key to the record in the database.

Authenticate to Twitter Using OAuth (continued)

The third and final stage of the OAuth process involves using the access token and access token secret and starting to interact with the Twitter API directly. These two values will not change as long as the user has authorized your Web application access, so you do not need to worry about the Reload button, bookmarks, or even manually typing in your URL. As long as the cookies remain intact, the user can access his profile with your Perl CGI script without interruption.

If you are only interested in using Twitter to securely authenticate users, you do not need to do anything with the Twitter API and their profile. A call to `validate_credentials` is prudent to ensure the access token is still valid; this method even returns the user's Twitter user ID, which you can treat as a unique identifier in your own code.

Authenticate to Twitter Using OAuth (continued)

- 19 Check the session hash for the 'access_token' and 'access_token_secret'.

Note: When found, the user is identified to be at stage three.

- 20 Type `$tw->access_token($sess{ 'access_token' });`;
21 Type `$tw->access_token_secret($sess{ 'access_token_secret' });`;
22 Change the stage two conditional test from `if` into `elsif`.

You do need to be careful using `Net::Twitter`, though. If a call fails, it triggers a kill signal that forces your CGI program to quit prematurely. For example, if the user suddenly removes their access token cookie, you will need your script to ignore the fatal error and attempt to re-authenticate the user. In Perl, you can temporarily bypass errors with the `eval` function:

```
eval { $tw->API_METHOD };
if ( $@ ) {
    warn "Error (ignored): $@";
    # Recover from the failure in API_METHOD.
}
```

This code instructs Perl to evaluate the block, but if any errors occur, they can be accessed with the special variable `$@`. This way, your program is given the opportunity to identify what failed, recover from the situation, and continue.

- 23 Type `my $results = eval{ $tw->verify_credentials() };`
24 Check if `$@` exists, and store it in `$err`.
25 Check if `$err->code` equals 401.
26 If so, call `&authorize_user()` and exit.

Note: Error code 401 means that the current access token is no longer valid. The user must start over at stage one: not authorized.

- 27 The API call succeeded, print the raw output of `$results` using `Data::Dumper`.
28 Save the Perl script.

```
14 );
15
16 my $sess = $cgi->cookie( 'sess' );
17
18 if ( $sess{ 'access_token' } && $sess{ 'access_token_secret' } ) {
19     $tw->access_token( $sess{ 'access_token' } );
20     $tw->access_token_secret( $sess{ 'access_token_secret' } );
21
22     warn "Getting Access token: " . $sess{ 'access_token' };
23     warn "Getting Access token secret: " . $sess{ 'access_token_secret' };
24 }
25 elsif ( my ( $request_token, $verifier ) = (
26     $cgi->param( "oauth_token" ),
27     $cgi->param( "oauth_verifier" ) ) ) {
28     $tw->request_token( $request_token );
29     $tw->request_token_secret( $sess{ 'request_token_secret' } );
30
31     $tw->request_access_token( verifier => $verifier );
32
33     warn "Getting Request token: " . $request_token;
34     warn "Getting Request token verifier: " . $verifier;
35     warn "Getting Access token: " . $tw->access_token;
36     warn "Getting Access token secret: " . $tw->access_token_secret;
37
38     my $cookie = $cgi->cookie( -name => 'sess', -value => {
39         access_token => $tw->access_token,
40         access_token_secret => $tw->access_token_secret,
41     } );
42
43     print $cgi->redirect(
44         -uri => "http://$ENV{ 'HTTP_HOST' }$ENV{ 'SCRIPT_NAME' }",
45         -cookie => $cookie );
```

```
46
47
48 my $results = eval { $tw->verify_credentials() };
49
50 if ( my $err = $@ ) {
51     if ( $err->code eq '401' ) {
52         &authorize_user();
53     }
54     else {
55         die $err->code . " " . $err->message . " " . $err->error . "\n";
56     }
57 }
58
59 print $cgi->header( 'text/plain' );
60 print Dumper( $results );
61
62 } elsif ( my ( $request_token, $verifier ) = (
63     $cgi->param( "oauth_token" ),
64     $cgi->param( "oauth_verifier" ) ) ) {
65     $tw->request_token( $request_token );
66     $tw->request_token_secret( $sess{ 'request_token_secret' } );
67
68     $tw->request_access_token( verifier => $verifier );
```

29 Load the Perl script in a browser.

- The Twitter OAuth authorization screen appears.

Note: Twitter requires you to allow the application to access your Twitter account.

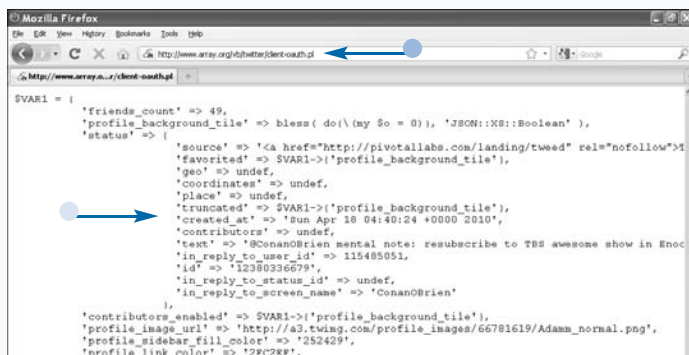
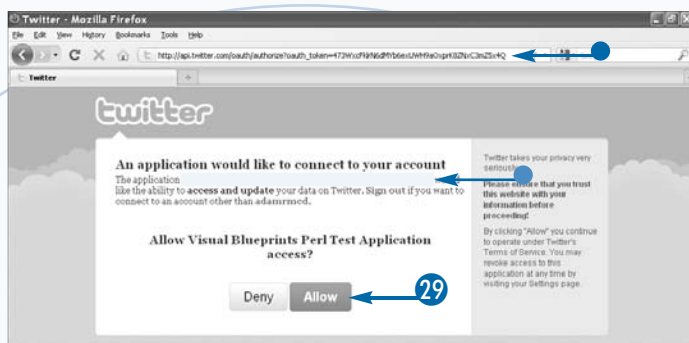
- Your registered application name appears.

Note: If you were not already logged into Twitter, you also see a username and password prompt on this screen.

30 Click Allow.

Twitter redirects you back to your Web site.

- Your Web site reloads.
- The browser displays the raw output of the `verify_credentials` API method using `Data::Dumper`.



Extra

The current structure of this program flows well enough as a single CGI script, handling all three stages of Twitter OAuth processing. However, its current structure appears odd, especially where the API call is executed; it actually finishes in the middle of the script.

Think of program design as writing a story: every good program has a clear beginning, middle, and end. It should always begin with its module initialization routines, as well as any one-time statements to set up its working environment. The middle is for the core program logic and functionality that matches the program's name and intention. It ends by displaying the final program output, and calling any general clean-up code. Your current program appears lopsided because the OAuth-handling code is actually initialization code that appears to seamlessly merge into middle code: the API call.

The solution is to convert your OAuth code into a custom Perl module that actually builds upon `Net::Twitter`. This new module can be initialized from any other Twitter CGI script, providing its OAuth functionality, yet not interfering with their core purpose. An example of the OAuth Twitter module, called `MyTwitter`, is described in the following section.

Create a MyTwitter Perl Module That Inherits Net::Twitter

You have a working Perl script that uses Net::Twitter, supports the special OAuth processing logic and can execute Twitter API functions. However, if you need to create multiple Perl CGI scripts, copying this code into each file is inefficient. Instead, you can move the OAuth processing code into a custom Perl module that *inherits* Net::Twitter, and use that as your Twitter initialization routine.

Perl supports an *object inheritance* feature, which allows you to take an existing Perl module's functionality, import it, build upon it, and store it as a new Perl module. In your case, Net::Twitter provides the majority of what you need for Twitter API connectivity, but it lacks the OAuth authentication process. Creating a new module that addresses the missing functionality, and inherits Net::Twitter's methods, is very easy:

Create a MyTwitter Perl Module That Inherits Net::Twitter

```
package MyTwitter;
use Net::Twitter;
@ISA = ( 'Net::Twitter' );
```

Simply assigning the parent module's name into an array called @ISA implies inheritance. You are announcing this MyTwitter module *is* a Net::Twitter module. Perl treats all code within MyTwitter.pm on the same level as Net::Twitter, as if the two exist as one file. Before transferring the OAuth code into MyTwitter.pm, you need to make one more modification. The original OAuth code used \$tw as its reference to Net::Twitter; your new module will need to maintain \$tw but source it differently. Because only core Perl scripts should initialize Perl modules, MyTwitter's subroutines will receive a ready-made \$tw module handle as an implicit first argument. The reference to your hybrid Net::Twitter module class is accessible as the first argument to its own subroutines through shift.

- 1 Open a blank text editor.
- 2 Type **package MyTwitter;**
- 3 Type **use CGI ':cgi';**
- 4 Type **use Net::Twitter;**
- 5 Type **@ISA = ("Net::Twitter");**
- 6 Type **1;**

```
1 package MyTwitter;
2
3 use CGI ':cgi';
4 use Net::Twitter;
5
6 @ISA = ( "Net::Twitter" );
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

- 7 Create an **init_session** subroutine.
- 8 Type **my \$tw = shift;**
- 9 Copy the OAuth code that handles stages two and three into **init_session**.
- 10 Type **return 0;** at the end of stage three.
- 11 Type **return 1;** at the end of **init_session**.

```
6 @ISA = ( "Net::Twitter" );
7
8 sub init_session {
9     my $tw = shift;
10
11     # OAuth code for stages two and three
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
247
```

- 12 Copy the `authorize_user` subroutine for stage one.
- 13 Type `my $tw = shift;`
- 14 Save the file as `MyTwitter.pm`.

```

22 $tw->request_access_token( verifier => $verifier );
23
24 my $cookie = cookie( -name => 'sess', -value => {
25     access_token => $tw->access_token,
26     access_token_secret => $tw->access_token_secret,
27 } );
28
29 print redirect( -uri => "http://$ENV{ 'HTTP_HOST' }$ENV{ 'SCRIPT_NAME' }",
30     -cookie => $cookie );
31
32 }
33 else {
34     authorize_user( $tw );
35 }
36
37 return 1;
38
39 }
40
41 my $tw = shift;
42
43
44
45
46
47
48
49
50
51
52
53 1;

```

- 15 Open a blank Perl script in a text editor.
- 16 Type `use MyTwitter;`
- 17 Type `MyTwitter->new` with the same arguments as `Net::Twitter` to initialize your new module.

Note: Remember, `MyTwitter` inherits `Net::Twitter`'s original `new` constructor subroutine.

- 18 Type `$tw->init_session() && exit 1;`
- 19 Create an `eval` block to execute API functions.
- 20 If there is a 401 error, call `$tw->authorize_user()` and exit.

```

#!/usr/bin/perl

use CGI ':cgi';
use CGI::Carp 'fatalToBrowser';
use MyTwitter;
use Data::Dumper;

my $tw = MyTwitter->new(
    traits => [ qw( API::REST OAuth ) ],
    consumer_key => '11111222233334444',
    consumer_secret => '0123456789abcdef0123456789',
);

$tw->init_session() && exit 1;

if ( my $err = $@ ) {
    die sprintf( "%s %s: %s\n", $err->code, $err->message, $err->error );
}

```

Extra

Your Perl CGI scripts can now use `MyTwitter` just like `Net::Twitter`, including your new OAuth method call, `$tw->init_session()`. Similar to Chapter 15 when you built your own authentication module, you can use this method's return code to determine if authorization failed, as in the case of OAuth at stages one or two. By returning code 1, your method instructs the normal program to stop.

The only disadvantage is that your Perl script still needs to be privy to error 401, and to call `$tw->authorize_user()`, just in case the current Twitter access token is unexpectedly revoked. To address that issue, you could take the contents of everything after the `eval` block and place it into `MyTwitter.pm` as a new subroutine: `catch_errors`. This is possible because the variable `$@` is globally accessible. Finally, you can simplify steps 18 and 19 to the following code:

```

eval { $tw->function( { ARGS } ) };
$tw->catch_errors();

```

Remember, you can always call multiple Twitter API functions within a single `eval` block. The `catch_errors` method automatically applies to any preceding function that throws an exception error.

Post a Twitter Status Update

You can post a status update through a Perl CGI script to your Twitter profile using Net::Twitter's update method. The simplest way to use it is to supply one parameter: the status message. Remember to use eval and trap any errors:

```
HASHREF = $tw->update( message );
```

You can also provide an anonymous hash into update to specify optional fields related to your post:

```
HASHREF = $tw->update( { status => message,
    OTHERARGS } );
```

Other optional fields you can specify using this format include `in_reply_to_status_id => statusid`, which allows you to refer to another tweet that you are replying. Also, if you have access to GPS coordinates, you can set the `lat` and `long` values and post a geo-encoded tweet. You can even try to identify what city or neighborhood

the user is in by using `$tw->reverse_geocode(lat, long)`. With this output, you can assign a new tweet to a location on a map, rather than just obtuse latitude and longitude coordinates.

The output of `update` will describe a summary of the status update. Generally speaking, if your update did not trigger an exception error with a corresponding `$@` variable, it was accepted. The most common error is code 403. This is used in multiple scenarios, including tweets that are longer than 140 characters, and multiple updates with the same text. Because you are using the OAuth method for authentication, the tweet source name and link used will be automatically set to your registered application name and URL. For more information, refer to the Twitter API documentation at <http://dev.twitter.com/doc/post/statuses/update/> or the Net::Twitter PerlDoc page.

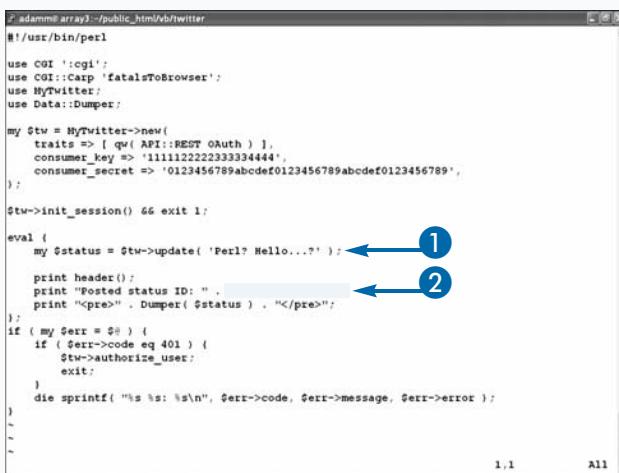
Post a Twitter Status Update

- 1 Type `my $status = $tw->update(message);`.
- 2 Use `$status->{ 'id' }`, to get the posted message status ID.

Note: You can always use `Dumper($status)` to see what keys are available in the `$status` hash reference.

Note: You can bundle multiple API calls within a single `eval` block. This way, you only need a single `$@` block to catch all API errors.

- 3 Save your Perl CGI script.
- 4 Open your Perl script in a Web browser.
 - Your status ID number.
 - The Data::Dumper output of `$status`.



```
#!/usr/bin/perl

use CGI ':cgi';
use CGI::Cisp 'katalaToBrowser';
use MyTwitter;
use Data::Dumper;

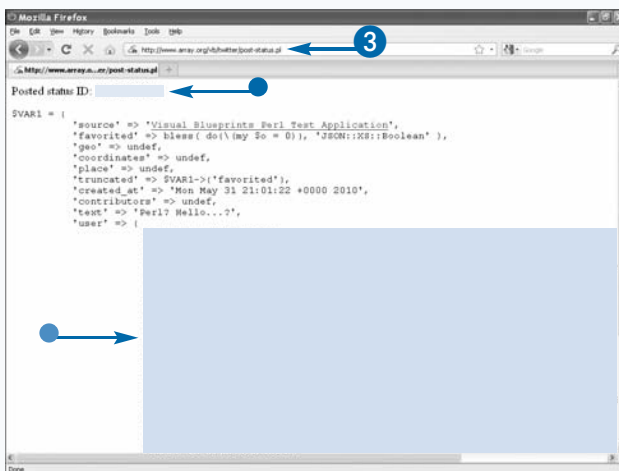
my $tw = MyTwitter->new(
    traits => [ qw( API::REST OAuth ) ],
    consumer_key => '1111222233334444',
    consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
);

$tw->init_session() && exit 1;

eval {
    my $status = $tw->update( 'Perl? Hello...?' );
    print header();
    print "Posted status ID: " .
    print "<pre>" . Dumper( $status ) . "</pre>";
};

if ( my $err = $@ ) {
    if ( $err->code eq 401 ) {
        $tw->authorize_user;
        exit;
    }
    die sprintf( "%s %s: %s\n", $err->code, $err->message, $err->error );
}

~
~
~
```



Retrieve a Twitter Timeline

You can retrieve various Twitter timelines using Net::Twitter and the Twitter API. This allows you to query all tweets posted by you, your friends, other users, or even the general public, and bring them into Perl CGI script:

```
ARRAYREF = $tw->user_timeline( { ARGS } );
```

You can use `user_timeline` without any arguments to retrieve the 20 most recent statuses posted by you into a Perl array reference. From here, you use `foreach` to iterate through each returned tweet, accessible by hash reference.

When searching by timeline, you can also set additional arguments. You can view a specific user's timeline by searching for the Twitter user profile using `user_id` to search by their Twitter User ID

Retrieve a Twitter Timeline

- 1 Type `my $list = $tw->friends_timeline({ ARGS });`.
- 2 Type `foreach my $tweet (@{ $list }) {`.
- 3 Use the hash ref `$tweet` to display information about each post in the timeline.
- 4 Type `}`.
- 5 Save your Perl CGI script.

Note: You can always use `Dumper($tweet)` to see what keys are available in the `$tweet` hash reference.

- 6 Open your Perl script in a Web browser.
 - The Web browser displays the requested timeline.

Note: You can experiment by using `user_timeline`, `home_timeline`, and `public_timeline`.

number, `screen_name` to search by screen name, or `id` to search by both user ID and screen name. You can also set the optional `since_id => statusid` to retrieve timeline events that occur later than that status ID number. You can also adjust the number of tweets returned by setting `count => number`, up to a maximum of 200, and view the next page of tweets with `page => number`.

Additional methods are available to query alternate timelines. You can specify `friends_timeline` to retrieve the latest tweets posted by you and the people you are following. The method `home_timeline` is identical to `friends_timeline`, except retweeted posts are also included. You can even use `public_timeline` to retrieve the latest tweets publicly posted by anyone, worldwide. Although these methods all allow for the same optional arguments, only `user_timeline` allows `id`, `user_id`, or `screen_name`.

```
#!/usr/bin/perl

use CGI ':cgi';
use CGI::Carp 'fatalToBrowser';
use MyTwitter;
use Data::Dumper;

my $tw = MyTwitter->new(
    traits => [ qw( API::REST OAuth ) ],
    consumer_key => '11111222233334444',
    consumer_secret => '0123456789abcdef0123456789'
);

$tw->init_session() 66 exit 1;

eval {
    my $list = $tw->friends_timeline();
    print header();
    print "<table>";
    foreach my $tweet ( @{ $list } ) {
        printf "<tr><td><img src='%s' height=48 width=48></td>" .
            "<td><b>%s</b><br>%s</td></tr>\n",
            $tweet->profile_image_url, $tweet->screen_name, $tweet->text;
    }
    print "</table>\n";
};
if ( my $err = $? ) {
    if ( $err->code eq 401 ) {
        $tw->authorize_user();
    }
}
```

Retrieve a List of Twitter Users You Follow

You can retrieve a list of Twitter users you follow using Net::Twitter and the Twitter API. This feature, according to the Twitter API, actually refers to this capability as a list of your Twitter *friends*. Therefore, a friend on Twitter is regarded as someone whose status you actively follow:

```
ARRAYREF = $tw->friends( { ARGS } );
```

You can use `friends` without any arguments to retrieve a list of the people you follow as a Perl array reference. You can view a specific user's friends list by searching for the Twitter user profile by using `user_id` to search by their Twitter User ID number, `screen_name` to search by screen name, or `id` to search by both user ID and screen name.

Because the returned list can be rather long, you can specify an optional `cursor => number` argument when

running your query. If you set it to `-1`, you retrieve about 100 users in your query. At the end the list, a `next_cursor` value recommends the new value to use to get the next 100 users. When there are no subsequent pages, `next_cursor` returns zero. The list is returned as an array reference variable, with each friend represented as a user hash reference within. For each user found, you can directly access information such as their last status update, profile image, homepage URL, user ID, screen name, location, language, and total number of friends and followers.

Twitter also provides additional friend-related API calls that allow you to create friend relationships, delete relationships, or even query the status of a pending friend request. For more information, refer to the Twitter API documentation at <http://dev.twitter.com/doc/post/statuses/friends/> or the Net::Twitter PerlDoc page.

Retrieve a List of Twitter Friends

- 1 Type `my $list = $tw->friends();`.
- 2 Type `foreach my $user (@($list)) {`.
- 3 Use the hash ref `$user` to display information about each friend.
- 4 Type `}`.
- 5 Save your Perl CGI script.

Note: You can always use `Dumper($user)` to see what keys are available in the `$user` hash reference.

- 6 Open your Perl script in a Web browser.
- The browser displays the list of friends (the users you follow).

```
#!/usr/bin/perl

use CGI ':cgi';
use CGI::Carp 'fatalToBrowser';
use MyTwitter;
use Data::Dumper;

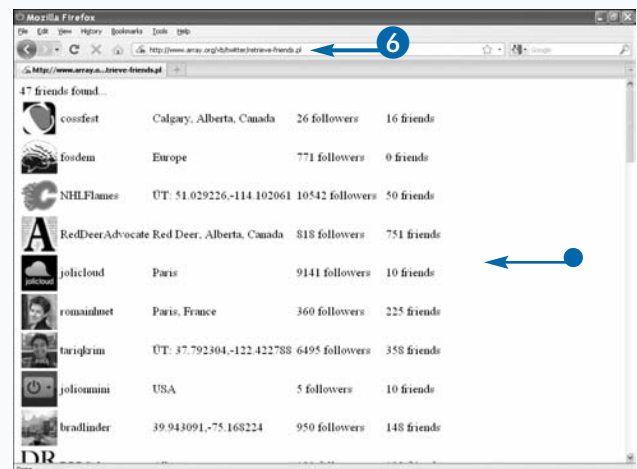
my $tw = MyTwitter->new(
    traits => [ qw( API::REST OAuth ) ],
    consumer_key => '11111222233334444',
    consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
);

$tw->init_session() && exit 1;

eval {
    my $list = $tw->friends();

    print header();
    printf "%d friends found...\n", $#($list) + 1;
    print "<table>";
    foreach my $user ( @$list ) {
        printf "<tr><td>img src='%s' height=48 width=48</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>\n",
            $user->profile_image_url,
            $user->screen_name,
            $user->location,
            $user->followers_count,
            $user->friends_count;
    }
    print "</table>\n";
};

if ( my $err = $? ) {
    if ( $err->code eq 401 ) {
```



Retrieve a List of Twitter Followers

You can retrieve a list of Twitter followers using Net::Twitter and the Twitter API. A follower is someone who is interested in your Twitter posts, and has selected your profile in their friends list:

```
ARRAYREF = $tw->followers( { ARGS } );
```

You can use `followers` without any arguments to retrieve a list of people who are following you as a Perl array reference.

When searching for followers, you can also set additional arguments. You can view a specific user's friends list by searching for their Twitter user profile using `user_id` to search by their Twitter User ID number, `screen_name` to search by screen name, or `id` to search by both user ID and screen name.

Retrieve a List of Twitter Followers

- 1 Type `my $list = $tw->followers();`.
- 2 Type `foreach my $user (@{ $list }) {`.
- 3 Use the hash ref `$user` to display information about each follower.
- 4 Type `}`.
- 5 Save your Perl CGI script.

Note: You can always use `Dumper ($user)` to see what keys are available in the `$user` hash reference.

Because the returned list can be rather long, you can specify an optional `cursor => number` argument when running your query. If you set it to `-1`, you retrieve about 100 users in your query. At the end of the list, a `next_cursor` value recommends the new value to use to get the next 100 users. When there are no subsequent pages, `next_cursor` returns zero.

The list is returned as an array reference variable, with each follower represented as a user hash reference within. For each user found, you can directly access information such as their last status update, profile image, homepage URL, user ID, screen name, location, language, and total number of friends and followers.

For more information, refer to the Twitter API documentation at <http://dev.twitter.com/doc/post/statuses/followers/> or the Net::Twitter PerlDoc page.

```
#!/usr/bin/perl

use CGI ':cgi';
use CGI::Carp 'fatalToBrowser';
use MyTwitter;
use Data::Dumper;

my $tw = MyTwitter->new(
    traits => [ qw( API::REST OAuth ) ],
    consumer_key => '11111222233334444',
    consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
);

$tw->init_session() or exit 1;

eval {
    my $list = $tw->followers();

    print header();
    printf "%d followers found...\n", $#{$list} + 1;
    print "<table>";
    foreach my $user ( @{$list} ) {
        printf "<tr><td><img src='%s' height=48 width=48 /><td>%s</td><td>%s</td><td>%s</td><td>%d followers</td><td>%d friends</td></tr>\n",
            $user->profile_image_url, $user->screen_name, $user->location, $user->language, $user->followers_count, $user->friends_count;
    }
    print "</table>\n";
};

if ( my $err = $? ) {
    if ( $err->code eq 401 ) {
```

- 6 Open your Perl script in a Web browser.
- The browser displays the list of followers.

Profile Image	Screen Name	Location	Followers	Friends
	liberprod	Paris	379 followers	2001 friends
	tangerfeer	Vancouver, Canada	121 followers	86 friends
	rveait		272 followers	331 friends
	pullingshots	Calgary, AB	80 followers	85 friends
	jkaczor	Calgary, AB	463 followers	1082 friends
	jerem	Paris	358 followers	223 friends
	moue86	Newyork	335 followers	1998 friends
	romainlnet	Paris, France	360 followers	225 friends
	zk	Boca Raton, Florida	12 followers	14 friends

```
HASHREF = $tw->search( { q => query, ARGS } );
```

the next page of tweets; `since_id => statusid` to restrict your search to statuses later than this status ID; and `until => yyyy-mm-dd` to only show tweets up to a specific date. Finally, a recent add-on to the Search API is the argument `result_type => format` where `format` is one of the following: `popular`, which returns only the most popular results; `recent`, which returns only the most recent results; or `mixed`, which combines popular and recent results. If you do not specify a `result_type`, it defaults to `mixed`. If you are interested, you can save a search query to the authenticated user's profile using the following code:

You can easily retrieve saved searches directly on the regular Twitter Web site, or with the following code:

```
HASHREF = $tw->saved_searches();
```

- 1 Add `API::Search` into `Net::Twitter`'s traits.

Note: Not specifying `lang` will return matching tweets in any language.

3 Type **foreach my \$tweet (@{ \$search->'results' }) {**.

- 4 Use the hash ref `$tweet` to display information about each returned post.

Note: You can always use `Dumper($tweet)` to see what keys are available in the `$tweet` hash reference.

5 Type }.

6 Load the `q` CGI parameter into `$query`.

- 7 Create an HTML form to prompt for a search term with a Submit button.

- 8 Check if `$query` is defined; if so, execute the search.

```

$ adamame array3 -f/public_html/vb/twitter/

7
8 my $tw = MyTwitter->new(
9     traits => { qw( API::Search My::Search ) },
10    consumer_key => '111112222333334444',
11    consumer_secret => '0123456789abcdef0123456789abcdef0123456789',
12 };
13
14 $tw->init_session() && exit 1;
15
16 eval {
17     print header();
18
19     my $search = $tw->search( {
20         q => $query, lang => 'en',
21     } );
22
23     print "Displaying %d results found for query '%s'...\n",
24           $#{$search} + 1, $search->{ 'query' };
25
26     print "<table>";
27     foreach my $tweet ( @{$search->{ 'results' } } ) {
28         print "<tr><td<img src='%s' height=48 width=48></td><td>%s</td><td>%s</td></tr>\n",
29               $tweet->{ 'profile_image_url' }, $tweet->{ 'from_user' },
30               $tweet->{ 'text' },
31               $tweet->{ 'created_at' };
32     }
33
34     print "</table>\n";
35 }
36
37 if ( my $err = $? ) {
38     if ( $err->code eq 401 ) {
39
40     }
41 }
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

adam@array3:~/public_html/vb/vbwriter
15
16 eval {
17     print header();
18     my $query = param( 'q' );
19
20     
21
22
23
24
25
26 if ( $query ) {
27     my $search = $tw->search(
28         q => $query, lang => 'en',
29     );
30
31     printf "Displaying %d results found for query '%s'...\n",
32           $#{$search}{ 'results' } + 1, $search->{ 'query' };
33
34     print "<table>";
35     foreach my $tweet ( @{$search}{ 'results' } ) {
36         printf "<tr><td<img src='%s' height=48 width=48></td><td><b>%s</b><br>%s<b>
37             </td></tr></td></tr>\n",
38               $tweet->{ 'profile_image_url' }, $tweet->{ 'from_user' },
39               $tweet->{ 'text' },
40               $tweet->{ 'created_at' };
41     }
42     print "</table>\n";
43 }
44
45 if ( my $err = $? ) {
46     if ( $err->code eq 401 ) {
47         $tw->authorize_user;
48     }
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

9 Type `my $page = param('page') || 1;`

Note: This sets `$page` to 1 only if the 'page' CGI parameter is not defined.

10 Type `page => $page.`

11 Add a link to open the next page of search results.

12 Save your Perl script.

13 Open the Perl script in a Web browser.

14 Type a search term.

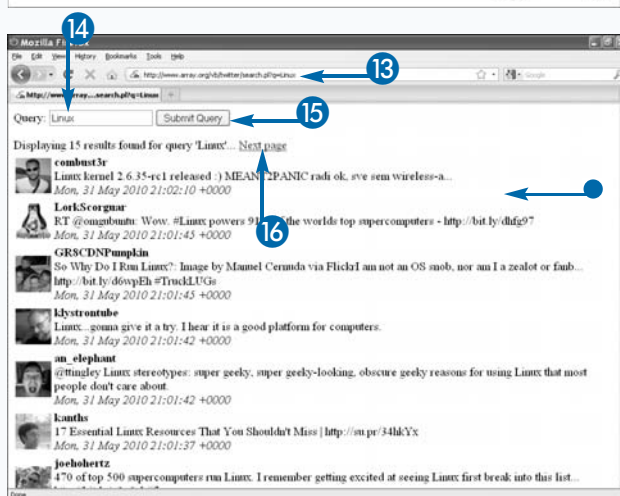
15 Click Submit Query.

- The search query results appear.

16 Click the Next page link.

The next page of search results appears.

```
adam@array3:~/public_html/vb/twitter
15
16 eval {
17     print header();
18     my $query = param( 'q' );
19
20     printf "<form method=get action='%s'>\n", $ENV{ 'SCRIPT_NAME' };
21     printf "Query: <input type=text name=q value='%s'>\n", $query;
22     printf "<input type=submit>\n";
23     printf "</form>\n";
24
25     if ( $query ) {
26         my $page = param( 'page' ) || 1;
27         my $search = $tw->search( {
28             q => $query, lang => 'en',
29         } );
30
31         printf "<a href='%s?q=%s&page=%d'>Next page</a>\n",
32             $ENV{ 'SCRIPT_NAME' }, $query, $page + 1;
33
34         print "<table>\n";
35         foreach my $tweet ( @{ $search->{ 'results' } } ) {
36             printf "<tr><td><img src='%s' height=48 width=48></td><td><b>%s</b><br><b>%s</b><br><b>%s</b><br></td></tr>\n",
37                 $tweet->{ 'profile_image_url' }, $tweet->{ 'from_user' },
38                 $tweet->{ 'text' },
39                 $tweet->{ 'created_at' };
40             print "</table>\n";
41         }
42     }
43 }
44
45 }
```



Extra

The Search API also features several calls to search on current trending topics on Twitter. This allows you to follow what people are talking about on Twitter, as the tweets happen, in real-time:

```
HASHREF = $tw->trends();
```

The `trends` method returns the top ten topics people are searching for, right now. You can also search by trending topics, which are popular topics that people are talking about in their tweets. You can see what is trending right now, or get a list of trends hour-by-hour in a day, or day-by-day in a week:

```
HASHREF = $tw->trends_current( { ARGS } );
```

```
HASHREF = $tw->trends_daily( { ARGS } );
```

```
HASHREF = $tw->trends_weekly( { ARGS } );
```

Each of these three methods accepts an optional `exclude => hashtags` argument. This alters your trend search to only list words and phrases that people are talking about. All trending hash tags are ignored. The daily and weekly methods can also search for trending topics in the past; simply specify the argument `date => yyyy-mm-dd` to see what was popular on that date.

Use the Twitter @Anywhere JavaScript API

You can use the Twitter @Anywhere JavaScript API to easily add Twitter functionality directly onto your Web site, without producing any Perl CGI code. The @Anywhere JavaScript API provides features such as auto-link user profiles, Hovercards, follow-me buttons, and a TweetBox. To import the @Anywhere base code, you can use the following JavaScript statement within the `<head>...</head>` tags of any HTML page, or in your header.shtml SSI page:

```
<script type="text/javascript" src="http://platform.twitter.com/anywhere.js?id=APIKEY&v=1"></script>
```

Be sure to set `APIKEY` with the appropriate value you received when you registered your Twitter application. To enable a specific @Anywhere feature, use the following JavaScript code anywhere in your HTML:

```
<span id="csstag"></span>
<script type="text/javascript">
  twttr.anywhere( function( T ) {
    T( "#csstag" ).feature( ARGS );
  }
</script>
```

The first line of the JavaScript never changes. The `twttr.anywhere` method loads the function `T` asynchronously on your page. You may specify multiple `T().feature()` calls within a single `twttr.anywhere` function block, as long as each call refers to a unique `csstag`.

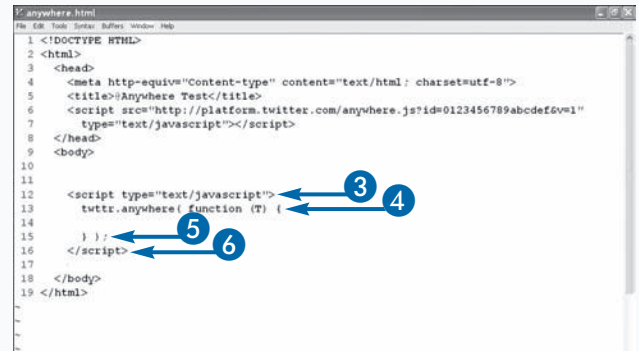
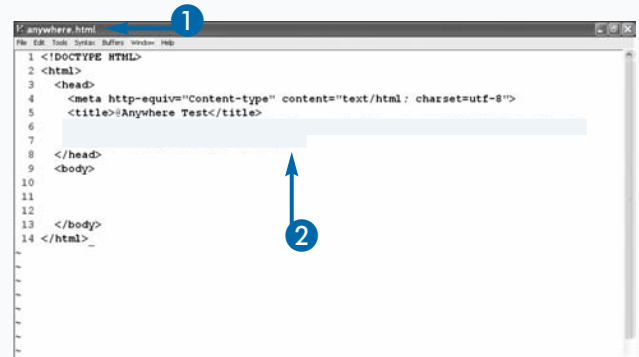
The `csstag` value refers to the CSS identifier where you want the feature to appear; note that this may be optional for some @Anywhere features. The `feature` refers to the API call you want to execute, and `ARGS` represents any API call arguments, if applicable.

Use the Twitter @Anywhere JavaScript API

- 1 Open an HTML Web page in a text editor.
- 2 Type `<script src="http://platform.twitter.com/anywhere.js?id=APIKEY&v=1" type="text/javascript"></script>`.

Note: If you do not yet have an API key, you must register your Web site as a Twitter application.

- 3 Type `<script type="text/javascript">`.
- 4 Type `twttr.anywhere(function(T) {`.
- 5 Type `};`.
- 6 Type `</script>`.



7 Type `<div id="NAME"></div>`.

Note: Make sure that the CSS tag ID value is unique.

Note: You can use almost any valid HTML tag, like `` and `<p id="NAME"></p>`.

8 Type `T("#NAME").function(ARGS);`.

Note: Not all @Anywhere functions require a CSS tag. In this case, use `T.function(ARGS);`

9 Save your HTML file.

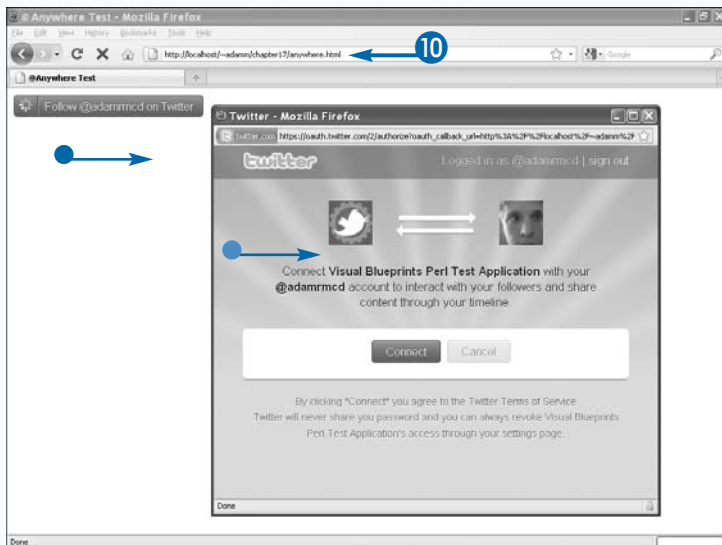
10 Open your HTML file in a Web browser.

- The browser displays the @Anywhere function.

11 Execute the function.

- A Twitter confirmation window appears.

```
anywhere.html
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4 <meta http-equiv="Content-type" content="text/html; charset=utf-8">
5 <title>@Anywhere Test</title>
6 <script src="http://platform.twitter.com/anywhere.js?id=0123456789abcdef6v=1"
7   type="text/javascript"></script>
8 </head>
9 <body>
10
11 <div id="follow-placeholder"></div>
12 <script type="text/javascript">
13   twttr.anywhere(function(T){
14     T("#follow-placeholder").followButton('adammcd');
15   });
16 </script>
17
18 </body>
19 </html>
```



Extra

When using @Anywhere functionality, Twitter may require a confirmation popup that confirms with the user that your Web site is going to interact with their Twitter account. Because your original HTML page will be used as a Callback URL after the user clicks the Allow button, you must make sure that the Callback URL you registered matches any Web page using the @Anywhere JavaScript SDK.

In contrast to this error message, only your Web site's domain name is actually used when comparing your HTML page to the Callback URL. This was a modification that Twitter applied shortly after launching @Anywhere because they realized that a single application may use the SDK on any URL, but the application must still only use a single domain name.

For more information, refer to the Twitter @Anywhere documentation at <http://dev.twitter.com/anywhere/begin/>.



You can use the Twitter Streaming API to actively monitor all Twitter activity. You can use a filter to only return tweets that match specific keywords, or simply return every tweet as it is posted to Twitter. The Streaming API permits you to open a persistent HTTP connection while your program is active, and Twitter will feed data through this connection in real-time.

Because Net::Twitter, and subsequently Perl, currently lack modular support for the Streaming API, you will have an easier time implementing it if you use basic authentication, using specific low-level Perl modules. Before you proceed with basic authentication, have a look at your copy of Net::Twitter's PerlDoc page. There is a chance that by the time you read this, support for the Streaming API will be added, and with it, support for using the API with OAuth.

There are three low-level Perl modules that you need to use: `Net::HTTP`, which allows for a persistent HTTP connection; `MIME::Base64` to support basic authentication; and `JSON` to decode the raw data into a Perl hash reference. Twitter says the Streaming API will try its best to keep your HTTP connection active indefinitely. Twitter recommends that you write your program to immediately reconnect if it becomes disconnected, but you should not abuse this if you encounter a legitimate error code. Specifically, they advise that if you receive an HTTP 200 error code, you should wait ten seconds before trying again. For each error you receive, double your wait time. Twitter does monitor errors on the Streaming API and if your program induces too many errors in a short period, you can be banned from the service.

For more information, refer to the Twitter @Anywhere documentation at http://dev.twitter.com/pages/streaming_api/.

streaming_api/.

- 1 Type `use Net::HTTP;`;
- 2 Type `use MIME::Base64;`
- 3 Type `use JSON;`;
- 4 Type `$| = 1;` to display data as it streams.
- 5 Type `my $http = Net::HTTP->new(Host => "stream.twitter.com");`;
- 6 Type `$http->write_request ('POST' => '/1/statuses/filter.json',);`;
- 7 Type `'Authorization' => encode_base64 ("username:password", ""),`;
- 8 Type `'Content-type' => 'application/x-www-form-urlencoded',`;
- 9 Type `'track=keywords',`;
- 10 Type `my ($code, $msg) = $http->read_response_headers();`;
- 11 Stop the program unless the response `$code` is 200.

adamm@array3:~/public_html/vb/twitter

```
1 #!/usr/bin/perl
2
3 use Data::Dumper;
4 use Net::HTTP;
5 use MIME::Base64;
6 use JSON;
```

$$8 \text{ \$} | = 1 ;$$

```
10 my $http = Net::HTTP->new( Host => "stream.twitter.com" );
```

12
13
14

```
17 my ( $code, $msg ) = $http->read_response_headers();
```

- 12 Type `$http->read_entity_body ($buf, 1024);`.
- 13 Append `$buf` onto a `$json` scalar.
- 14 Process the raw JSON data through a new `decode` subroutine.
- 15 Use `from_json` to decode the raw JSON input into a hash ref.
- 16 Examine the contents of `$data` with `Data::Dumper`.
- 17 Save the Perl script.

```

adammi array3 -/public_html/vb/twitter
5 use MIME::Base64;
6 use JSON;
7
8 $i = 1;
9
10 my $http = Net::HTTP->new( Host => "stream.twitter.com" );
11 $http->write_request(
12     'POST' => '/1/statuses/filter.json',
13     'Authorization' => encode_base64( "username:password", "" ),
14     'Content-type' => 'application/x-www-form-urlencoded',
15     'track=linux';
16 );
17
18 my ( $code, $msg ) = $http->read_response_headers();
19 die "$code: $msg" unless ( $code eq 200 );
20
21 my $json = "";
22
23 while ( 1 ) {
24     my $buf;
25     my $n = $http->read_entity_body( $buf, 1024 ); ← 12
26     die "read failed: $?" unless defined $n;
27     last unless $n;
28     $json .= $buf; ← 13
29     $json =~ s/((.*))$&decode( $1 )/ge;
30 }
31
32 sub decode ( $ ) {
33     my $input = shift;
34     my $data = from_json( $input ); ← 15
35     print Dumper( $data ), "\n"; ← 16
36 }

```

- 18 Run the Perl script in a Terminal window.
 - Twitter starts to feed the filtered data to your browser from the Streaming API.
 - The program waits for the next tweet in the stream.
- 19 Press `Ctrl+C` to stop the program.

```

adammi array3 -/public_html/vb/twitter
[adammi@twitter]$ ← 18
$VAR1 = (
  'source' => 'Ca href="http://dlvr.it" rel="nofollow">dlvr.it</a>',
  'favorited' => bless( do{ (my $o = 0) }, 'JSON::XS::Boolean' ),
  'coordinates' => undef,
  'geo' => undef,
  'place' => undef,
  'truncated' => $VAR1->{'favorited'},
  'created_at' => 'Mon May 31 21:32:04 +0000 2010',
  'contributors' => undef,
  'text' => 'Linux $!vm Re: Is it possible to bypass LVM and mount contained part
tion directly? http://dlvr.it/1L7Yd',
  'in_reply_to_user_id' => undef,
  'user' => (
    'friends_count' => 21,
    'profile_background_tile' => $VAR1->{'favorited'},
    'profile_sidebar_fill_color' => '000000',
    'contributors_enabled' => $VAR1->{'favorited'},
    'profile_link_color' => '008484',
    'profile_image_url' => 'http://a1.twimg.com/profile_images/927999440/
angsana-new-duck_normal.png',
    'profile_sidebar_border_color' => 'a8c7f7',
    'created_at' => 'Mon May 24 23:47:50 +0000 2010',
    'time_zone' => undef,
    'favourites_count' => 0,
    'geo_enabled' => $VAR1->{'favorited'}
  ),
  'id' => '15133200645',
  'in_reply_to_status_id' => undef,
  'in_reply_to_screen_name' => undef
);

```

Extra

Basic authentication is not very secure. You may want to consider manually implementing `Net::OAuth` with the Streaming API, or you can use your personal access token and secret value that you received when you first registered your Web site as a Twitter application. Throughout this chapter, you have only used `OAuth`; however, in this case you have no choice, as `Net::Twitter`, which makes `OAuth` easier to implement, does not support the Streaming API.

Four services are made available by the Streaming API:

SERVICE URL	DESCRIPTION
<code>/1/statuses/filter.json</code>	A filtered feed of all Twitter posts, by matching keyword.
<code>/1/statuses/firehose.json</code>	An unfiltered raw feed of all Twitter posts.
<code>/1/statuses/sample.json</code>	A random sample feed of Twitter posts, useful for testing purposes only.
<code>/1/statuses/retweet.json</code>	A raw feed of all Twitter retweets.

Remember to review the documentation for each API call at <http://dev.twitter.com/doc/get/statuses/name>.

Accept a File for Upload

By allowing your users to upload files, you can expand the types of services that your Web site offers by writing a Perl CGI script that accepts the uploaded data, processes it, and produces some sort of result. To allow users to upload files, you must first create an HTML form that prompts the user for the file to upload, and then create a Perl CGI script that accepts the upload using the CGI library.

To accept files, your opening HTML form tag must specify a special *encoding-type* attribute called "multipart/form-data", after which, all data fields that are sent will be specially formatted for complex data payloads. This allows you to accept relatively large files, and to maintain support for legacy form input fields.

Within the actual form, you must add a "file" input field. Its syntax is exactly the same as the other form-input fields such as "text", "password", and "submit",

Accept a File for Upload

except this field triggers additional functionality in the Web browser that allows the user to browse their personal hard drive and select a file to upload:

```
<form method="post" method="URL"
      enctype="multipart/form-data">
<input type="file" name="field"><input
type="submit">
```

In your Perl script, the CGI library provides a built-in way to decode the "multipart/form-data" protocol by way of the upload method. You provide the following code:

```
$handle = $cgi->upload( field );
$data .= $_ while( <$handle> );
```

The CGI library does not restrict the maximum file size, but you should enable an upper limit to prevent any abuse of Web server resources. To configure your Perl script to apply an upper limit, type the following code before initializing the CGI library:

```
$CGI::POST_MAX = bytes;
```

1 Open a Perl CGI script in a text editor.

2 Type **enctype='multipart/form-data'** in the opening HTML form tag.

Note: Make sure the method is "post". This encoding-type is incompatible with the "get" method.

3 Type **<input type=file name='upload-file'>** inside the HTML form.

4 Check if the CGI upload-file parameter exists.

5 Type **my \$upload = \$cgi->upload ('upload-file');**.

6 Type **my \$data = ""**.

7 Type **\$data .= \$_ while (<\$upload>);**

```
1 #!C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use Data::Dumper;
6
7 my $cgi = new CGI;
8
9 print "Content-type: text/html\n\n";
10 printf "<form method=post action='%s' enctype='multipart/form-data'>"
11     $ENV{ 'SCRIPT_NAME' };
12 print "<input type=file name='upload-file'>"
13 print "<input type=submit value='Upload'>";
14 print "</form>";
15
```

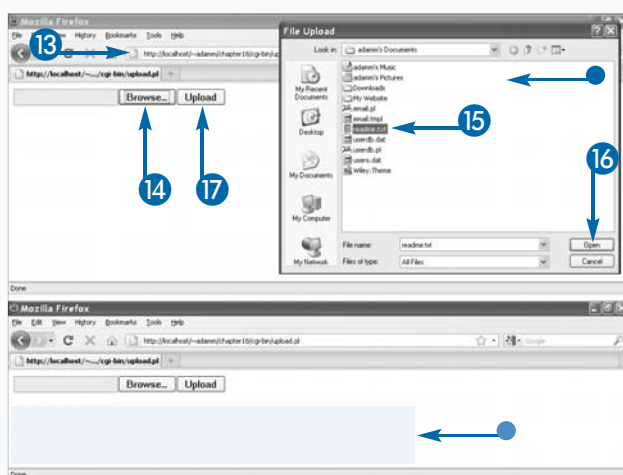
```
16 if ( $cgi->param( 'upload-file' ) )
17     my $upload = $cgi->upload( 'upload-file' );
18     my $data = "";
19
20     $data .= $_ while ( <$upload> );
21 }
```

- 8 Type `my $info = $cgi->uploadInfo ($upload)`;
- 9 Type `length($data)` to get the number of bytes received.
- 10 Type `$info->{ 'Content-Type' }` to access the file's MIME type.
- 11 Type `$info->{ 'Content-Disposition' }` to access the file's original filename.
- 12 Save the Perl CGI script.

```

1 #!C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalsToBrowser';
5 use Data::Dumper;
6
7 my $cgi = new CGI;
8
9 print "Content-type: text/html\n\n";
10 print "<form method=post action='$s' enctype='multipart/form-data'>";
11 $ENV{'SCRIPT_NAME'};
12 print "<input type=file name='upload-file'>";
13 print "<input type=submit value='Upload'>";
14 print "</form>";
15
16 if ( $cgi->param( 'upload-file' ) ) {
17     my $upload = $cgi->upload( 'upload-file' );
18     my $info = $cgi->uploadInfo( $upload );
19     my $data = "";
20
21     $data .= $_ while ( <$upload> );
22
23     print "File upload received:<br>\n";
24     print "<div>Bytes: " . length( $data ) . "</div>";
25     print "<div>Content-Type: " . $info->{ 'Content-Type' } . "</div>";
26     print "<div>Content-Disposition: " . $info->{ 'Content-Disposition' } . "</div>";
27 }
  
```

- 13 Load the CGI script in a Web browser.
- 14 Click Browse.
- The File Upload window appears.
- 15 Select a file.
- 16 Click Open.
- 17 Click Upload.
- The Perl script receives the file in memory.



Apply It

You can actually use the technique described here for any file type. Given your Web site's objectives, you may want to limit the types of files that will be accepted by filtering on the file's MIME type.

TYPE THIS

```

if ( $info->{ 'Content-Type' } =~ /^image\\/ ) {
    # Process the uploaded image file.
}
  
```



RESULTS

You can add this code immediately after calling the `$cgi->upload(field)` statement. Once implemented, you can ensure that only files whose MIME type begins with "image/" will be accepted.

To save the file to disk, you need to have Perl use binary mode on your output file. After calling `$handle = $cgi->upload(field)`, you can save it to disk.

TYPE THIS

```

open( SAVE, ">filename" );
binmode( SAVE );
print SAVE $_ while ( <$handle> );
close( SAVE );
  
```



RESULTS

The uploaded file is saved to disk using binary mode.

Open an Image with Image::Magick

Perl can use the ImageMagick graphics library to open, create, manipulate, and save image files to disk. ImageMagick is accessible from the Perl module Image::Magick. If you are on Linux, you can install the package `perlmagick`. If you are on Windows, you can download the installation program from www.imagemagick.org/script/binary-releases.php#windows. Be sure to select the installation option, Install PerlMagick for ActiveState Perl. After installing ImageMagick, reboot your Web server to find the Perl module:

```
use Image::Magick;
my $image = new Image::Magick;
my $err = $image->Read( filename );
```

Once the `$image` handle has been initialized and an image file is read, you can access ImageMagick's methods through the `$image` handle.

Open an Image with Image::Magick

- 1 Open a new Perl script in a text editor.
- 2 Type `use Image::Magick;`
- 3 Type `my $image = new Image::Magick;`

- 4 Type `my $err = $image->Read(filename);`
- 5 Type `die $err if $err;`

Note: The function `die` will kill your program at that point; however, if you want to continue but also display a warning, then use `warn`.

Apply It

After opening the image with `Read`, you can access the image's attributes with `Get`.

TYPE THIS:

```
my ( $w, $h ) = $image->Get( 'width',
    'height' );
```



RESULTS:

The width and height of the image are written into the variables `$w` and `$h`.

```
P: image.pl
File Edit Tools Syntax Buffers Window Help
1 #!C:/Perl/bin/perl.exe
2
3 use Image::Magick;
4
5 my $image = new Image::Magick;
6
~
~
~
~
~
~
~
~
~
~
```

```
P: image.pl
File Edit Tools Syntax Buffers Window Help
1 #!C:/Perl/bin/perl.exe
2
3 use Image::Magick;
4
5 my $image = new Image::Magick;
6
7 my $err = $image->Read( "myimage.jpg" );
8 die $err if $err;
9
~
~
~
~
~
~
~
~
~
~
```


Resize or Crop an Image with Image::Magick

Once you open an image using the Image::Magick Perl module, you can use the corresponding `$image` handle to begin manipulating the image data. For an online gallery application, this can be particularly useful as you may want to convey the image in multiple display sizes: a thumbnail for browsing the picture in a list, a scaled image suitable for viewing the image in a Web browser, and the original image for downloading and printing.

You may choose to resize the image dynamically in your Web site, as the image is requested by the user's Web browser in real-time, but this will take away CPU resources from your Web server, possibly resulting in slower performance. Instead, you should

resize the image in a separate process or server, shortly after receiving the original image, and store the resized copies of the original image on disk. The method for resizing is `Resize`:

```
$image->Resize( width => width, height => height );
```

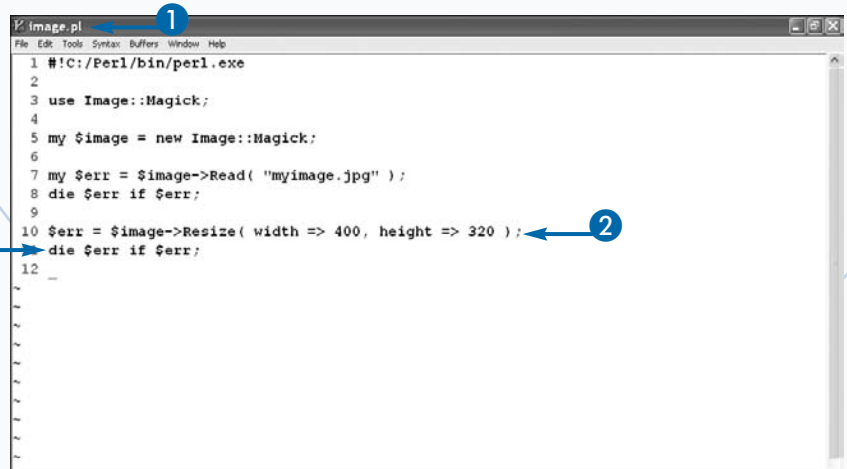
The `resize` function does not have a built-in aspect ratio lock. This means that you are free to change the ratio between the width and height, but if you only change one value, the other remains the same. If you want to lock it, you must calculate the width and height values.

ImageMagick also allows you to crop an image to a new size, but keep the original aspect ratio. The method for cropping is `Crop`:

```
$image->Crop(x => x, y => y, width => width, height => height);
```

Resize or Crop an Image

- 1 Open a Perl script with an image loaded into an `$image` handle.
- 2 Type `$err = $image->Resize(width => width, height => height);`.
- 3 Type `die $err if $err;`.



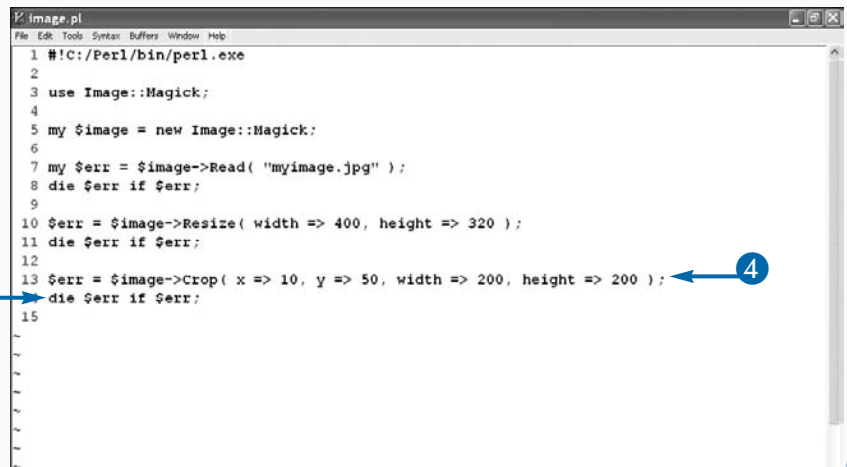
```

1 image.pl
2 #!C:/Perl/bin/perl.exe
3 use Image::Magick;
4
5 my $image = new Image::Magick;
6
7 my $err = $image->Read( "myimage.jpg" );
8 die $err if $err;
9
10 $err = $image->Resize( width => 400, height => 320 );
11 die $err if $err;
12

```

- 4 Type `$err = $image->Crop(x => x, y => y, width => width, height => height);`.
- 5 Type `die $err if $err;`.

Note: Either function can accept a geometry value to specify coordinates, for example, `$image->Crop(geometry => "XxY+W+H");`.



```

1 image.pl
2 #!C:/Perl/bin/perl.exe
3 use Image::Magick;
4
5 my $image = new Image::Magick;
6
7 my $err = $image->Read( "myimage.jpg" );
8 die $err if $err;
9
10 $err = $image->Resize( width => 400, height => 320 );
11 die $err if $err;
12
13 $err = $image->Crop( x => 10, y => 50, width => 200, height => 200 );
14 die $err if $err;
15

```

Manipulate an Image with Image::Magick

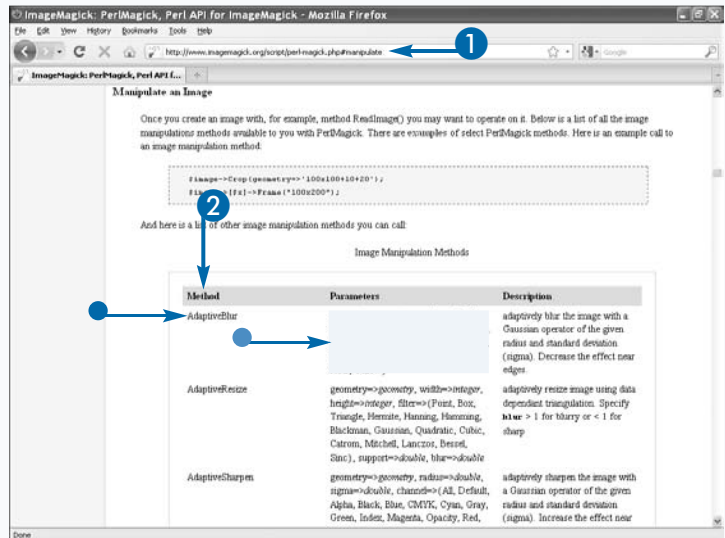
ImageMagick supports dozens of image effect filters that are accessible by your Perl script through the Perl Image::Magick module. Manipulation filters include the ability to blur, sharpen, rotate, scale, implode, explode, swirl, negate, trim, wave, and tint any image. You can also draw lines, circles, squares, or rectangles on images, and even add colored text using a fancy font. Additional advanced color manipulation commands are also available. This includes the ability to segment an image, apply a median filter, alter the color map, and enhance or reduce the image brightness and contrast. ImageMagick also supports some built-in manipulation *recipes* that will add some graphical flare to an image using multiple manipulation commands. This includes adding an ornamental frame, even making the image look like an oil painting, a charcoal drawing, or a picture.

Manipulate an Image with Image::Magick

- 1 Browse to the ImageMagick image-manipulation methods Web page.
- 2 Select a filter method.
 - The method's name.
 - The method's parameters.

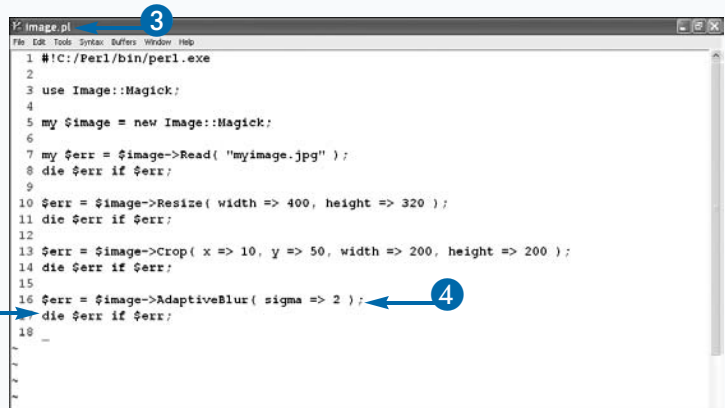
Extra

You can find a complete list of available filters and arguments at www.imagemagick.org/script/perl-magick.php#manipulate. Once you identify a filter method you want to apply, the arguments summarize what the filter accepts, but the Web site does not explain which ones are required. You can find additional documentation about each manipulation filter at www.imagemagick.org/Usage/.



- 3 Open a Perl script with an image loaded into an `$image` handle.
- 4 Type `$err = $image->method (parameters);`.
- 5 Type `die $err if $err;`.

When you run your program, the script executes your chosen manipulation function.



Save an Image to Disk

After modifying an image with ImageMagick, you can save it to disk. If you choose, you can change the format of the file simply by changing the output filename extension:

```
$err = $image->Write( "filename.jpg" );
die $err if $err;
```

If there is a problem writing the image to disk, `$err` is populated with the error message. If the file already exists, ImageMagick automatically appends a number to the end of the filename — for example, *filename-0.jpg*, *filename-1.jpg*, and so on. No message is returned on `$err` if this happens. However, one way to avoid this problem is to test if the file already exists, remove it, and then save it:

```
my $filename = "filename.jpg";
unlink( $filename ) if ( -e $filename );
$err = $image->Write( $filename );
die $err if $err;
```

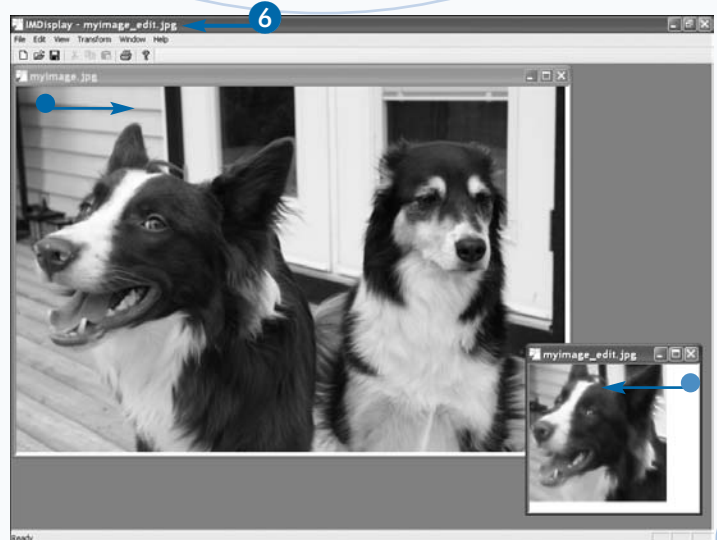
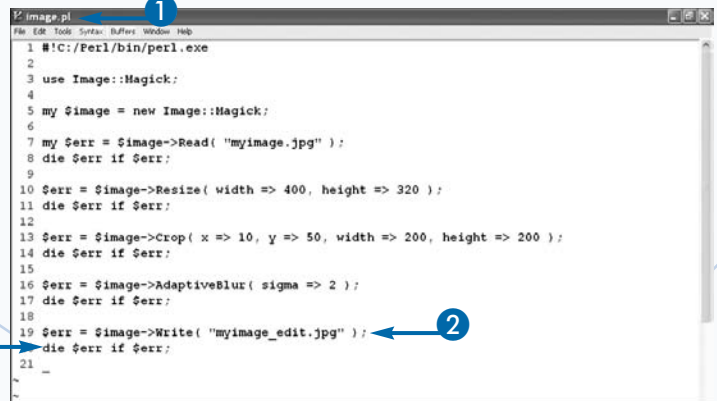
After saving the file once, the image data is still available in `$image`. This is useful in case you want to apply a modification to the image, but you want to save the current image to disk prior to doing so.

Image::Magick supports the most common Internet image file formats, such as GIF, JPEG, and PNG, as well as dozens of less-common formats. If you want to review a complete list of formats that are available for your installation of ImageMagick, use this command:

```
print join( ' ', $image->QueryFormat() );
```

Save an Image to Disk

- 1 Open a Perl script with an image loaded into an `$image` handle.
- 2 Type `$err = $image->Write (filename);`
- 3 Type `die $err if $err;`
- 4 Save your Perl script.
- 5 Execute your Perl script.
- 6 Open an image viewer.
 - The original image file.
 - The edited image file.



Display a Dynamic Image to the Browser

You can construct a program that takes advantage of the CGI library's file-upload functionality, and the Image::Magick module's manipulation functionality to produce a CGI Web page that accepts an image, manipulates it through command links, and displays the dynamic output of the image back to the user. Because there are multiple display states that can be presented to the end-user, this program needs to be aware of what state the user is in, based upon the information provided from the previous state. The trick is to code each state in your CGI script *backward*. Naturally, when a user first visits this Web page, they should only see an upload prompt, but that upload prompt may be preceded by the uploaded image, if the image is already stored on disk.

Display a Dynamic Image to the Browser

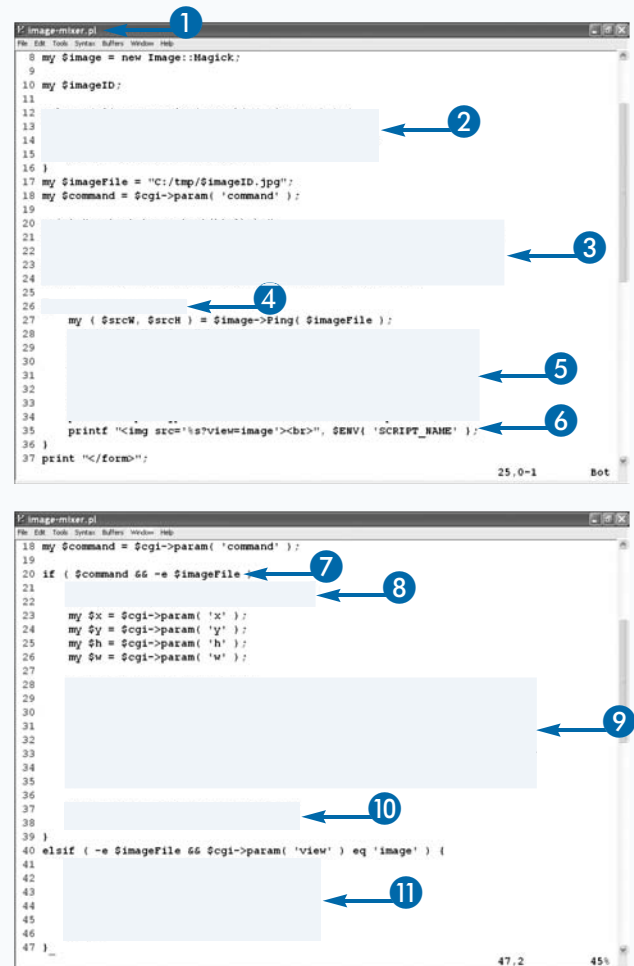
- 1 Open a Perl CGI script that uses the CGI and Image::Magick modules.
- 2 Assign and retrieve a unique cookie that will track the user's image name on disk.
- 3 Display the HTML form to upload a new image.
- 4 Check if the image exists.
- 5 Display some input parameters to modify it.
- 6 Display the image by calling this Perl CGI script with a special CGI parameter.
- 7 Check if an image exists and if a command was requested.
- 8 Read the image file into ImageMagick.
- 9 Perform the selected command.
- 10 Write the image file back to disk.

Note: Undo functionality can be provided by sequentially backing up the current image just before executing the Image::Magick function.

- 11 If you use the view CGI parameter, load the image file and display it as the Perl CGI script's output.

You can verbalize your code like this: If you have an image, display it; otherwise, if you are receiving an image, save it; otherwise, just display the upload prompt. Hence, backwards. This idea is extended to the next level, displaying various commands when displaying the image, but preceding the display with the execution of the actual commands.

One challenge in creating this CGI script is that you want multiple people to be able to use the CGI script at the same time, but you also want each user's uploaded files isolated to the person who uploaded them. The last thing you should do is pass a literal filename as a CGI parameter to the file that is being manipulated. Instead, as a rudimentary solution that provides moderate security, you can assign a user a multi-randomized cookie, and then use this cookie value to reference the actual image filename.



- 12 Check if the user is uploading a file.
- 13 Store the uploaded data into a temporary file.
- 14 Type **binmode(SAVE);** to write the temporary file to disk in binary mode.
- 15 Type **print SAVE \$_ while (<\$upload>);** to save the image to the temporary file.
- 16 Open the temporary file with **Image::Magick**, and save it as a cookie-image JPEG.
- 17 Save the CGI script.

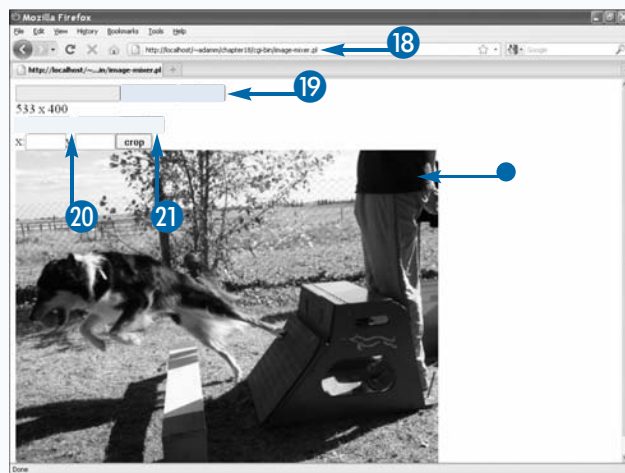
```

18 my $command = $cgi->param( 'command' );
19
20 if (
21     ...
22     ...
23     ...
24     # Save uploaded data into a temporary file
25     open( SAVE, ">$tmpFile" );
26     binmode( SAVE );
27     print SAVE $_ while( <$upload> );
28     close( SAVE );
29
30     # Use Image::Magick to read the temporary file, and save it as a JPEG.
31     # Remove any existing file.
32     unlink( $imageFile ) if ( -e $imageFile );
33
34     # Remove the temporary file.
35     unlink( $tmpFile );
36
37 }
38
39 elsif ( $command eq "-e $imageFile" ) {
40     my $err = $image->Read( $imageFile );
41     die $err if $err;
42     my $x = $cgi->param( 'x' );
43     my $y = $cgi->param( 'y' );
44     my $h = $cgi->param( 'h' );
45     my $w = $cgi->param( 'w' );
46
47     if ( $command eq "resize" ) {

```

- 18 Open the CGI script in a Web browser.
- 19 Upload an image.
 - The image appears in the Web browser.
- 20 Type new height and width values.
- 21 Click Resize.

The image displays at the new size you specified.



Extra

For simplicity, this program referenced the image filename using a literal, yet obfuscated, value. Nonetheless, this is a weak security model. A more robust solution is to add in a user-authentication layer, as described in Chapter 15, and tie the user's identifier to a table that restricts the user to viewing and manipulating only those images that they uploaded.

While this is a fair amount of work, you need to be extremely diligent in protecting your user's personal data on your Web site. If your CGI program is sloppy or incomplete, someone could gain access to view not only another user's images, but even system files stored locally on your Web server.

This is actually a risk under this current model. One way to circumvent any attempts at accessing other files is to sanitize the cookie's value by removing potentially dangerous characters, such as dots or slashes. Because you use the literal cookie as your image filename, a dot or a slash could result in an attacker probing your system for private files. Simply reject a cookie that contains anything other than numbers or letters, and re-assign it to a new value. See Chapter 22 for more information on securing your CGI code.

You can use the Perl module `Authen::Captcha` to create images that you can use as Captcha tests on your Web site. The acronym stands for Completely Automated Public Turing test to tell Computers and Humans Apart. This is a very useful way to limit portions of your Web site to human-users, and to generally stop computer-users, such as automated programs browsing the Web, from passing. See Chapter 9 for more information on installing modules.

When initializing `Authen::Captcha`, you need to provide two directories. The first, `data_folder`, indicates a path where it can store its database. This should be a directory that is not visible from an Apache URL. The second, `output_folder`, indicates a path where you want `Authen::Captcha` to write its Captcha images. This must be within a public URL directory.

Implement an Image Captcha Test

This module requires a Perl CGI script that you write in two parts. The first part requires you to call the method `generate_code`, which creates the Captcha image, writes it to disk in an output folder, and returns the `md5sum` hash of the correct answer. Your program is responsible for displaying the image and generating the HTML form that prompts the user for the answer. In the actual form, you must pass the `md5sum` hash as a hidden input value.

The second part is called from the first by way of submitting the form. It must call the method `check_code` with the `md5sum` hash and the user's answer as arguments. If the answer matches the `md5sum` hash, the method returns the value 1, indicating a pass. Any number other than 1 is a failure. Your program needs to be able to identify the passing result and only then decide whether to allow the user to continue.

- 1 Open a new Perl CGI script in a text editor.
- 2 Type **use Authen::Captcha;**
- 3 Type **my \$captcha = new Authen::Captcha;**
- 4 Type **\$captcha->data_folder (*secure-path*);**
- 5 Type **\$captcha->output_folder (*public-path*);**

6 Type `my $md5sum = $captcha->generate_code(number);`.

Note: This number argument indicates the total number of characters that should appear in the Captcha image.

- 7 Display the `$md5sum.png` image.
- 8 Use this script as the HTML form's action attribute.
- 9 Pass `$md5sum` as a hidden form variable.
- 10 Prompt for the user's answer.

```

1  #!C:\Perl\bin\perl.exe
2
3  use CGI;
4  use Authen::Captcha;
5
6  my $cgi = new CGI;
7  my $captcha = new Authen::Captcha;
8  $captcha->data_folder( 'c:/tmp' );
9  $captcha->output_folder( '.' );
10

```

```

1  #!C:\Perl\bin\perl.exe
2
3  use CGI;
4  use Authen::Captcha;
5
6  my $cgi = new CGI;
7  my $captcha = new Authen::Captcha;
8  $captcha->data_folder( 'c:/tmp' );
9  $captcha->output_folder( '.' );
10
11 print "content-type: text/html\n\n";
12
13
14 print "<form method=post action='";
15 print "<form method=post action='";
16 print "<form method=post action='";
17 print "<form method=post action='";
18 print "</form>";

```


- 11 Check if the `md5sum` and answer CGI variables were provided.
- 12 Type `my $result = $captcha->check_code (answer);`.
- 13 Check if `$result` is 1, indicating a pass.
- 14 Otherwise, display a failure message.

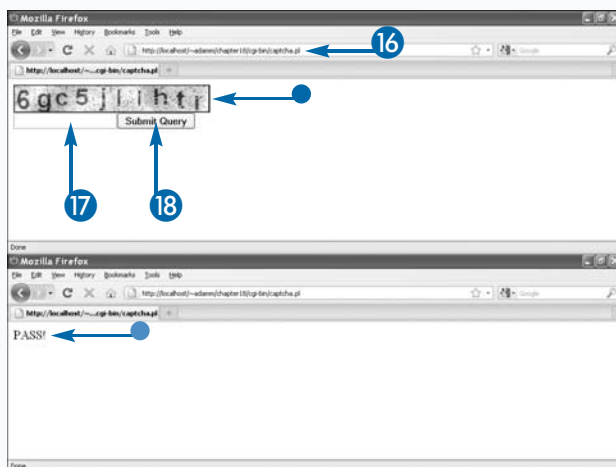
Note: For security purposes, do not provide an explanation as to why the user's answer failed the test. The `$result` code does provide other error levels, but you should only use these for debugging.

- 15 Save the Perl script.
- 16 Load the CGI script in a Web browser.
 - The Captcha prompt appears.
- 17 Type an answer.
- 18 Click Submit Query.
 - The failure or success message appears.

```

1  #!/usr/bin/perl
2  use strict;
3  use warnings;
4  use CGI;
5  my $cgi = new CGI;
6  my $captcha = new Authen::Captcha;
7  $captcha->data_folder( 'c:/tmp' );
8  $captcha->output_folder( '.', ' ' );
9
10
11 print "content-type: text/html\n\n";
12 if (
13     [
14         $cgi->param('md5sum'),
15         $cgi->param('answer')
16     ]
17 )
18 {
19     my $result = $captcha->check_code( $cgi->param('answer') );
20     if ($result == 1)
21     {
22         print "PASS!\n";
23     }
24     else
25     {
26         print "FAIL!\n";
27     }
28 }
29 else
30 {
31     my $md5sum = $captcha->generate_code( 10 );
32     print "<img src='$md5sum.png'>\n";
33     print "<form method='post' action='$_SERVER{SCRIPT_NAME}'>";
34 }

```



Extra

Each time `Authen::Captcha` generates an image or validates a response, it checks its data file for previously generated images, and it automatically deletes all used Captcha images.

The images produced by `Authen::Captcha` are relatively rudimentary, but they do serve the purpose well enough. If you are protecting an area of your Web site that is relatively important, you may want to consider increasing the number of characters in the image, decreasing the expiry timeout, or even opening up the JPEG image file with `Image::Magick` and adding in your modifications to make it more visually complex.

A Captcha test does not need to be only an image. Some commercial products are capable of producing a sound-based test, or even a word-based math problem test.

For example, you might prompt your users with the message, “What is thirty plus seventeen minus five?” and request that they type their answer as words, not numbers.

A Perl CGI script that demonstrates this type of test is provided on this book's Web site.

Produce an Image Gallery

You can produce a simple image gallery using Perl to showcase your photos online. Such a program does not have to be overly complex. If all you want to do is showcase your vacation photos to your family using a public URL, then a Perl CGI script is more than sufficient.

The gallery is this example accepts images for upload, and presents them in three display modes: thumbnail, review, and original. The thumbnail mode allows you to preview all images in your gallery using a smaller image (around 140 x 140 pixels), displaying all available images on one page. Clicking the thumbnail allows you to view the image in more detail, using a resolution more suitable to your Web browser, possibly at 500 x 300 pixels. Clicking that image displays the original image, using its original resolution.

Produce an Image Gallery

- 1 Open a Perl script in a text editor.
- 2 Import the CGI, Image::Magick, and HTML::Template modules.
- 3 Define a location for the images, relative to the script.
- 4 Define the program's stages: upload, display sized image, and display thumbnail list.
- 5 Display the template as program output.
- 6 Identify the uploaded file's name, excluding the extension.
- 7 Define the locations for the temporary upload, and the original, sized, and thumbnail files.
- 8 Save the uploaded data into the temporary file.
- 9 Use Image::Magick to read the temporary file and write it as the original image. This converts all uploads into JPEG images.
- 10 Use Image::Magick's `Resize` method to produce the sized and thumbnail versions.

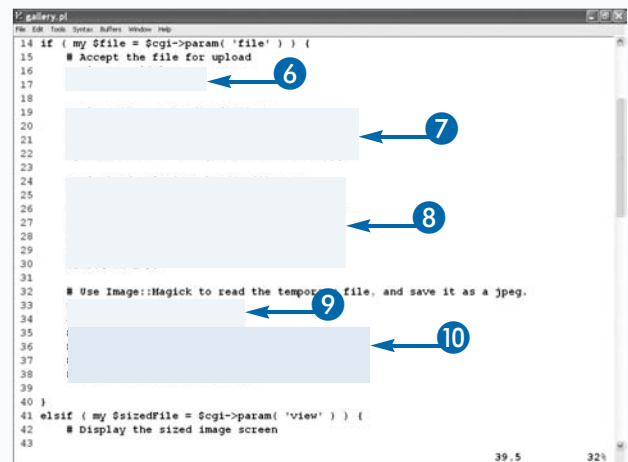
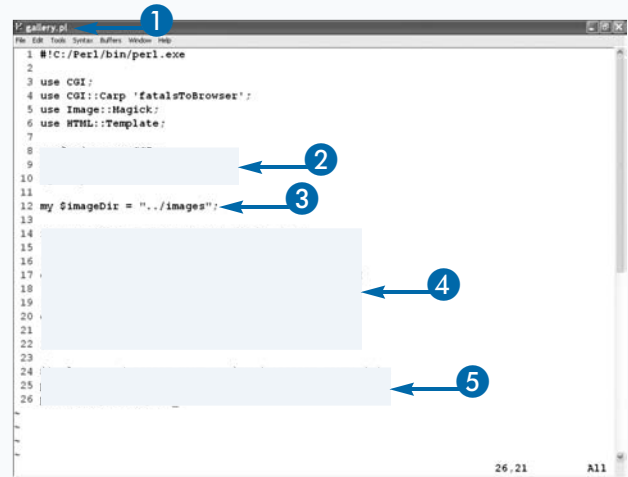
Note: Because you will have multiple template files, you should only declare the `$tmpl` handle for now.

You can also add in a slide show feature, or include a way to browse to the next and previous image while reviewing an image.

When receiving an uploaded image, the program converts its size right away, and redirects the Web browser back to the previous page, reloading the thumbnails so the new image is visible. This process is fairly easy, and avoids the problem of clicking the Reload button immediately after uploading an image, which effectively uploads it again. After the image is uploaded and processed, you add the following line of code:

```
print "Location: $ENV{ 'HTTP_REFERER' }\n\n";
```

This location line, when used prior to `Content-type: text/html`, redirects the user's browser back to the referring location. This effectively divides your program between processing code and displaying code.



- 11 Delete the temporary file, and redirect the user to the previous URL.
- 12 Identify the original filename from the sized filename.
- 13 Open the `gallery_review.tmpl` template.
- 14 Assign the sized and original files as parameters.

```

31
32 # Use Image::Magick to read the temporary file, and save it as a jpeg.
33 $image->Read( $tmpFile );
34 $image->Write( $origFile );
35 $image->Resize( width => 500, height => 300 );
36 $image->Write( $sizedFile );
37 $image->Resize( width => 140, height => 140 );
38 $image->Write( $thumbFile );
39
40
41
42
43 elsif ( my $sizedFile = $cgi->param( 'view' ) ) {
44     # Display the sized image screen
45
46
47
48
49
50
51
52 }
53 else {
54     # Display image preview (thumbnails) screen.
55
56 }
57
58 $tmpl->param( SCRIPT_NAME => $ENV{ 'SCRIPT_NAME' } );
59 print "Content-type: text/html\n\n";
60 print $tmpl->output;
  
```

- 15 Compile a list of thumbnails.
- 16 For each thumbnail, identify the sized filename.
- 17 Push the thumbnail and sized files into the `$list` array reference.
- 18 Open the `gallery_thumbnails.tmpl` template.
- 19 Assign the list of thumbnails as a parameter.
- 20 Save the Perl script.

```

45 my $origFile = $sizedFile;
46 $origFile =~ s/-sized/-orig/;
47
48 $tmpl = HTML::Template->new( filename => 'gallery_review.tmpl',
49     die_on_bad_params => 0 );
50 $tmpl->param( sizedFile => $sizedFile );
51 $tmpl->param( origFile => $origFile );
52
53 else {
54     # Display image preview (thumbnails) screen.
55     my @thumbFiles = <$imageDir/*-thumb.jpg>;
56     my $list = [];
57
58     foreach my $thumbFile ( @thumbFiles ) {
59
60
61
62
63
64
65
66
67
68
69 $tmpl->param( fileList => $list );
70 }
71
72 $tmpl->param( SCRIPT_NAME => $ENV{ 'SCRIPT_NAME' } );
73 print "Content-type: text/html\n\n";
74 print $tmpl->output;
  
```

Apply It

The hard-coded values used in the steps for the thumbnail and review images are not scaling consistently. When resizing an image, maintain the height and width ratio between the original image and the resized copy; this prevents distortion of the image.

TYPE THIS

```

my ( $w, $h ) = $image->Get( 'width', 'height' );
my ( $w2, $h2 );
if ( $w > $h ) {
    $w2 = 140; $h2 = int( $h * $w2 / $w );
} else {
    $h2 = 140; $w2 = int( $w * $h2 / $h );
}
$image->Resize( width => $w2, height => $h2 );
  
```



RESULTS

The code sets the larger image dimension to be the maximum value, 140, and cross-multiplies by the original height and width to find the shorter dimension that keeps the same image aspect ratio.

continued



Produce an Image Gallery (continued)

You may want to add some technical information about the picture while the user is reviewing the scaled copy. Many digital cameras produce JPEG images that contain additional data in EXchangeable Image File (EXIF) format. While Image::Magick cannot read EXIF content, there are Perl modules that can. One such module is called Image::ExifTool. You can install it over CPAN or ActiveState PPM, as described in Chapter 9. The easiest way to use it, and to review the available EXIF fields in an image, is to couple it with Data::Dumper:

```
use Image::ExifTool ':Public';
use Data::Dumper;
my $info = ImageInfo( $filename );
print Dumper( $info );
```

If you already know what fields you are interested in, you can bypass importing Data::Dumper and call the fields' names directly:

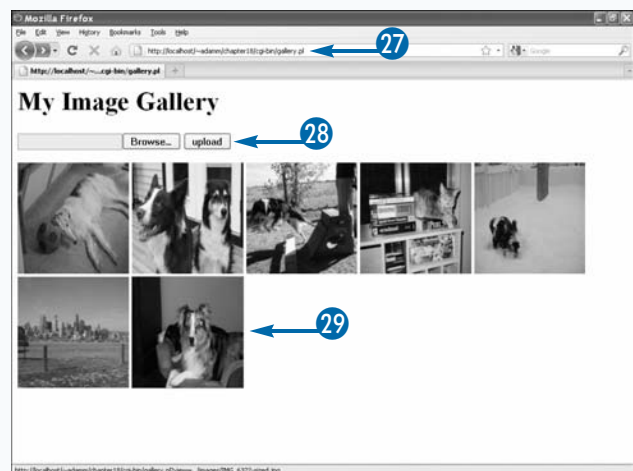
```
print "Camera model: $info->{ 'Model' }\n";
print "Exposure: $info->{ 'ShutterSpeed' }\n";
print "Picture Date: $info->{ 'CreateDate' }\n";
```

Depending on your camera model, a lot of data may be returned. Fortunately, even though Image::Magick does not understand EXIF, it does maintain the data structure if you modify a JPEG file that already contains the data.

For more information on Image::ExifTool, see its PerlDoc page. For more information on EXIF in general, go to <http://en.wikipedia.org/wiki/EXIF>.

Produce an Image Gallery (continued)

- 21 Open the `gallery_thumbnails.tpl` template file.
- 22 Insert the file upload HTML code.
- 23 Display the loop of thumbnail images, linked to the sized image.
- 24 Open the `gallery_review.tpl` template file.
- 25 Display the sized file, linked to the original file.
- 26 Save both templates.
- 27 Open the gallery CGI script in a Web browser.
The list of thumbnails in the gallery appears.
- 28 Browse for an image and click Upload.
The new image's thumbnail appears.
- 29 Click a thumbnail image.

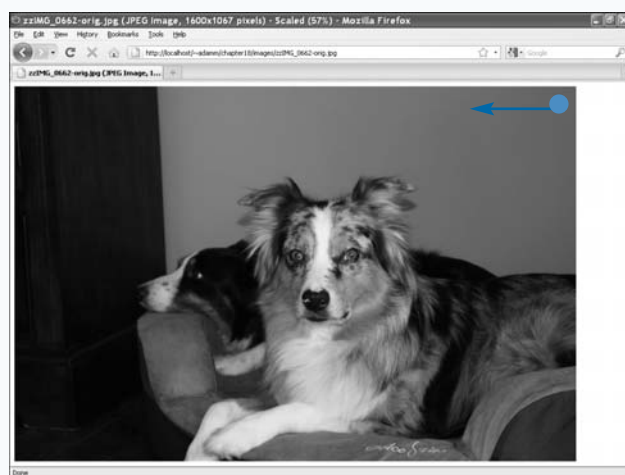
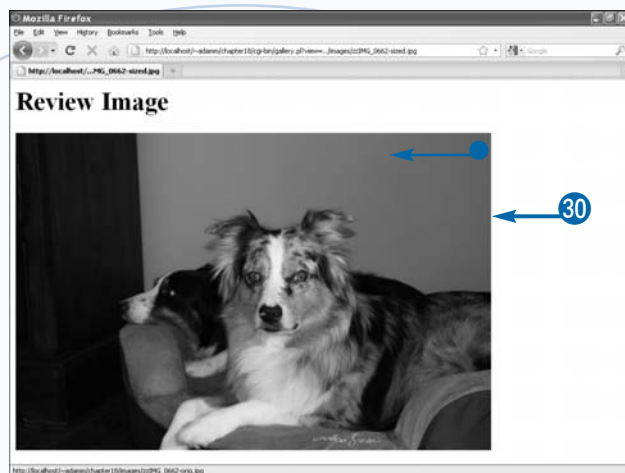


- The browser displays the resized image.

30 Click the image.

- The original image appears.

Note: Even though the URL refers to the original JPEG file, the Web browser may still try to scale the image to fit in the window.



Extra

After you set up a basic image gallery, you can start adding in simple image-editing tools, such as image rotation, cropping, or even image modification effects. You may also want your users to be able to apply a title to the image, or delete the image entirely from the gallery. If you add in these types of gallery editing features, you also need to add in some form of user authentication. This will prevent anyone from accessing your Web site and editing your images.

In this example, you are displaying the images directly in the HTML output, rather than through a Perl CGI script wrapper as demonstrated earlier in this chapter. If you want to build your gallery to be completely private, you should consider wrapping the images in a Perl CGI, and then leverage your authentication model in the same code.

The single-image upload window is not entirely convenient, especially if you have several new images to post. You could create multiple file-upload form fields, but a more clever solution would be to specify a local directory on your Web server. Simply FTP your images in bulk into this directory, and instruct your Gallery Web site to scan them.

Introducing AJAX

AJAX, or Asynchronous JavaScript and XML, is a development technique that uses client-side JavaScript to communicate using XML through to a server-side component, independent of the initial HTML download of a Web page. AJAX has the benefit of being able to change an already-loaded Web page dynamically, without needing to download and render an entirely new page just to display new content.

Normally, after an initial page load, when the user clicks a button on a Web page, the browser loads the URL link assigned to the button, and the entire Web page refreshes. After implementing AJAX, that button click actually opens up a connection to the Web server, independent of the

Web browser's display window, performs a new request, downloads some new information, and dynamically displays it on the current Web page. This makes AJAX Web pages appear and respond much faster than traditional HTML, at least after the initial page load.

The individual parts of AJAX have been available since the mid-1990s, but this particular combination and usage was not common until 2005 when Google showcased their Gmail and Google Maps applications. These sites are just regular Web pages with HTML and JavaScript, but with the creative execution of AJAX by Google, they became dynamic, fast-loading Web applications.

The Purpose of AJAX

AJAX eliminates the need to reload a Web page each time the user clicks a link and interacts with it. Prior to AJAX, regardless of the complexity of the request made by a user, the entire Web page needed to be downloaded and rendered again. For a site with a lot of graphics and content, this was slow and inefficient.

AJAX helps alleviate this problem by allowing the Web developer to only update content that has changed, leaving the rest of the page untouched from the initial Web page download.

AJAX Process Flow

Several key JavaScript features come together to form the AJAX experience. The two main ones are the asynchronous Web server calls, and the dynamic updates to the browser's currently rendered Web page.

Asynchronous Interaction

Because JavaScript is already a Web browser-based programming language, it is aware of how the end-user is interacting with the Web site. This awareness is limited to the code already delivered to it by the Web server each time the Web page loads. With the advent of asynchronous interaction functions in JavaScript, such as `XMLHttpRequest`, Web developers can code JavaScript to query the Web server for information outside of the initial Web page download. This query performs just like a traditional HTTP GET or POST, except the Web browser does not reset its display. JavaScript is completely in charge of making the request to the Web server, receiving the request, and parsing it for useful information to enhance the end-user's experience on the Web site.

Early AJAX implementations used XML-formatted content as a communication language between the JavaScript and the CGI. This made sense because standard HTML is a graphical rendering language, but JavaScript is only concerned with transferring raw information. However, XML is overly verbose and can be inefficient at sending large amounts of data. Even the JavaScript functions designed to parse and navigate XML are inefficient and convoluted. A leaner communication language called JSON has since become a popular alternative to XML. When reading JSON, it appears like literal JavaScript variable assignments. This makes it very easy to implement JSON in JavaScript. In fact, in Chapters 16 and 17 you will find that both Facebook and Twitter are producing services that deliver dynamic content using JSON instead of XML.

Dynamic Web Page Updates

The end-user experience is enhanced by the use of the JavaScript property `innerHTML`. Text written to this property inside of an HTML object automatically appears in the Web browser at the placement of that tag. Nearly every tag in HTML can be assigned an identifier which can be located using JavaScript. By assigning any HTML tag with an identifier, JavaScript can populate content within the tag dynamically on the current Web page. For example, the following code is invisible when the Web browser first renders the page:

```
<span id=output></span>
```

By using JavaScript to locate the `output` tag by its identifier, you can inject new content into the Web page using the tag's `innerHTML` property:

```
var outputObj = document.getElementById(
    'output' );
outputObj.innerHTML = "New content for the span
tag!";
```

Because AJAX is already pulling in new data from the Web server, it is a logical tie-in for it to use `innerHTML` to update the Web browser with that data.

Implementing AJAX

When implementing an AJAX solution, you need to be aware of special considerations on both the Web browser and server.

Web Browser Support

Because AJAX relies primarily on JavaScript technologies, implementing it actually becomes more difficult as each Web browser can implement these specialized JavaScript functions differently. The Web developer's goal is to produce a single Web site that works in all browsers; however, JavaScript complicates that goal because each browser implements JavaScript with its own extensions and enhancements. The process of Web browser identification and JavaScript function probing leads to code that is two or three times longer than it should be, just because of each Web browser's idiosyncrasies.

Fortunately, developers have produced several JavaScript libraries dedicated to simplifying the JavaScript development effort, and reducing the time spent implementing AJAX. One such library is called jQuery (www.jquery.com). You only need to follow jQuery's common execution syntax to perform individual JavaScript and AJAX tasks. Since each Web browser has slightly different implementations of JavaScript, jQuery automatically identifies the browser in use and executes the correct browser-specific JavaScript code.

Web Server Support

From the perspective of Apache or any other Web server, it does not need to worry about AJAX-specific Web traffic. The `XMLHttpRequest` function produces HTTP queries in the same fashion as the regular browser; Apache just fulfills the request to the CGI program that is servicing the requested URL.

Perl CGI Support

Perl has no built-in functions that specifically handle AJAX, JavaScript, or even HTML. Because these are all languages and techniques that are executed on the Web browser, and Perl is only designed to be executed on the Web server, your Perl CGI scripts can only use `print` to relay these language commands to your user's Web browser. This means that you are free to implement AJAX manually, if you so choose, but you must do this by coding JavaScript and HTML *within* Perl `print` statements.

Fortunately, there are some helper modules that automate the construction of AJAX and JavaScript in Perl, before it is sent to the Web browser. One of these modules, `CGI::Ajax`, is demonstrated throughout this chapter.

Introducing CGI::Ajax

You can use the Perl module CGI::Ajax to simplify asynchronous Web site activity produced with Perl CGI scripts. You do not necessarily need CGI::Ajax to add AJAX functionality under Perl, but it does help.

CGI::Ajax provides a connection between JavaScript and Perl, allowing you to transmit data between both execution

environments. This is useful in case you have a feature that you want to execute in JavaScript, but it requires data that must be generated by Perl on a Web server. CGI::Ajax bridges that gap so the two programming environments can talk seamlessly to each other.

Installation

CGI::Ajax is a third-party Perl module that is not a part of the standard Perl distribution. You must manually install it on all Web servers and workstations that you use to run Perl CGI scripts. The process to install it is just like any other third-party module, using any tool described in Chapter 9. CGI::Ajax does have one dependency module, Class::Accessor.

Class::Accessor is a tool that Perl module developers use to automatically generate simple methods in Perl modules that can get and set module-specific variables, called accessors. Additional documentation is available for both modules by way of the PerlDoc program.

Requirements

Before you can implement CGI::Ajax, you need to reorganize how your Perl CGI script flows.

Registered Perl Subroutines

CGI::Ajax requires you to register your Perl subroutines as JavaScript functions. This is one of the best features of CGI::Ajax as it provides for you a convenient and automated way to link JavaScript activity with Perl functionality. When JavaScript makes a request for one of your subroutines, you need to be careful about how code outside of your registered subroutine is executed. You must avoid code that is automatically executed outside of the requested subroutine that prints anything as output, as this affects how CGI::Ajax transmits your Perl output back to JavaScript.

Your registered Perl subroutines receive data directly from JavaScript in the form of an array. It in turn is expected to return one or more values, also as an array, which populates a predetermined HTML tag as identified by its CSS ID field.

Centralized HTML Code

CGI::Ajax does not function well if your Perl script prints HTML at various points, scattered throughout its logic. This is because your Perl CGI actually runs multiple times at various states. The first state is general output; it is the normal code requested when the page first loads. The second state contains the asynchronous calls back to the script; you do not want any uncontrolled HTML to be displayed, as this may adversely affect the field that CGI::Ajax is trying to update.

CGI::Ajax recommends that you create a subroutine, typically named `show_html`, which gets called from within its controller method, aptly named `build_html`. Because CGI::Ajax always knows what state it is acting in, it can decide if it needs to display the HTML, depending on whether the user is newly visiting your CGI, or performing an asynchronous request.

If you are already separating HTML from Perl using another library such as HTML::Template (as described in Chapter 13), then you are already on the right track. In fact, HTML::Template integrates very easily with CGI::Ajax because your template's HTML output has already been disconnected from the logic and functions within your Perl CGI script.

Process Workflow

CGI::Ajax makes executing asynchronous calls easy because it wraps around your Perl CGI scripts, providing a predetermined workflow to both JavaScript and Perl.

Initialize CGI::Ajax

Your Perl CGI script can initialize CGI::Ajax by importing its module and creating a module handle with the `new` method. In order for CGI::Ajax to work, you must register at least one JavaScript function-to-Perl subroutine reference when initializing the module:

```
use CGI::Ajax;
my $ajax = CGI::Ajax->new( jsFunction => perl_
    subroutine );
```

The CGI library is also required, but it is not used directly within CGI::Ajax until you define your HTML output.

Your HTML code needs to be controlled by CGI::Ajax so it only displays at specific times. The easiest way to do this is to create a subroutine specific to displaying HTML, and *reference* it in CGI::Ajax's `build_html` method:

```
print $ajax->build_html( $cgi, \&show_html );
```

The reference to your HTML subroutine is vital as you do not want to actually execute it when calling `build_html`, but instead pass a reference so that `build_html` can call it when it is ready. CGI::Ajax also expects a module reference to the CGI library, which is required because it produces proper CGI headers and can interact with other CGI functions on your behalf.

CGI::Ajax expects that your HTML code is properly formatted in HTML 2.0 format. This means that it contains a proper `<html><head></head><body>...</body></html>` format structure. CGI::Ajax parses your HTML and injects its JavaScript code at the correct point.

Your HTML code should have some way of making JavaScript function calls, triggering your registered Perl subroutines, based upon some sort of user event or activity.

CGI::Ajax Builds the Output

When your CGI script is loaded into a Web browser, CGI::Ajax begins to work and delivers a large chunk of JavaScript code to the user automatically. This is to

facilitate the asynchronous JavaScript calls later so that they work regardless of the Web browser.

CGI::Ajax takes the liberty of also posting your CGI headers to the Web server. If you do not want this functionality, you can always call `$ajax->skip_header(1)`. You may also want to send additional arguments that would normally be sent to `$cgi->header`. This can happen automatically if you provide them as a third argument to `build_html`.

Call Perl Asynchronously

Depending on the Web browser event you want to watch for, your HTML code should reference a JavaScript function registered to a Perl subroutine. Two arguments are required for the JavaScript function: the first is the input variables or CSS ID fields that you want to send to Perl, and the second is the CSS ID field or fields that you want to be updated to display Perl output:

```
Your Email: <input type=text id=email
onchange="validate_
name(['email'], ['result'])">
Validity check: <span id=result></span>
```

Execute Perl Subroutines

Your Perl subroutine automatically receives as input parameters the data stored within HTML fields in the Web browser. For example, if the user types **john@foo.com** into the `'email'` text input field, Perl automatically receives that data as the first parameter of input to its subroutine registered to the `validate_name` JavaScript function.

Update the Web Page

When your Perl subroutine ends, it is expected to return something back to CGI::Ajax. This output is automatically placed within another HTML tag somewhere on the Web page. For example, if the user's e-mail address passes the Perl subroutine's validity check, it may return "Looks good!" This is automatically placed within the tag with the `'result'` ID attribute.

Add CGI::Ajax into Your Perl CGI Scripts

You can add CGI::Ajax into your Perl CGI scripts to simplify AJAX communication between JavaScript and Perl. The CGI::Ajax module injects JavaScript code automatically into your CGI script's output, providing an easy-to-use communication bridge between JavaScript code and Perl code. JavaScript now has access to Perl functionality and database content only found on your Web server:

```
use CGI;
use CGI::Ajax;
my $cgi = new CGI;
my $ajax = CGI::Ajax->new( Registered
    Functions );
```

Somewhere in the middle of your CGI script, you should create an HTML subroutine that describes your CGI's output. Because CGI::Ajax needs to inject a large amount of JavaScript to provide the AJAX functionality, CGI::Ajax must correctly place your script's HTML within its own code. As a result, you cannot just print HTML code; you must make it available within a subroutine called by CGI::Ajax when it needs it.

The last line of code in your Perl CGI script should be the `build_html` method, with the CGI library's handle, as well as a *reference* to your HTML subroutine:

```
print $ajax->build_html( $cgi, \%show_html );
```

There is no need to specify `$cgi->header()` prior to `build_html`, as CGI::Ajax takes care of that for you.

Add CGI::Ajax into Your Perl CGI Scripts

- 1 Open a Perl CGI script in a text editor.
- 2 Type `use CGI::Ajax;`
- 3 Type `my $ajax = new CGI::Ajax(jsFunction => \%perl_subroutine);`
- 4 Create a placeholder `perl_subroutine` subroutine.

Note: CGI::Ajax requires at least one registered Perl subroutine.

- 5 Return dynamic data.

- 6 Create a `show_html` subroutine.
- 7 Type `return <<EOF;`
- 8 Call the placeholder `jsFunction` function.
- 9 Create a `span` tag to display the Perl output.

Note: The exchange of Perl and JavaScript variables is described in more detail in the section, "Call Perl Subroutines through JavaScript." For now, you just need a placeholder, as CGI::Ajax requires at least one subroutine.

- 10 Type `EOF`.

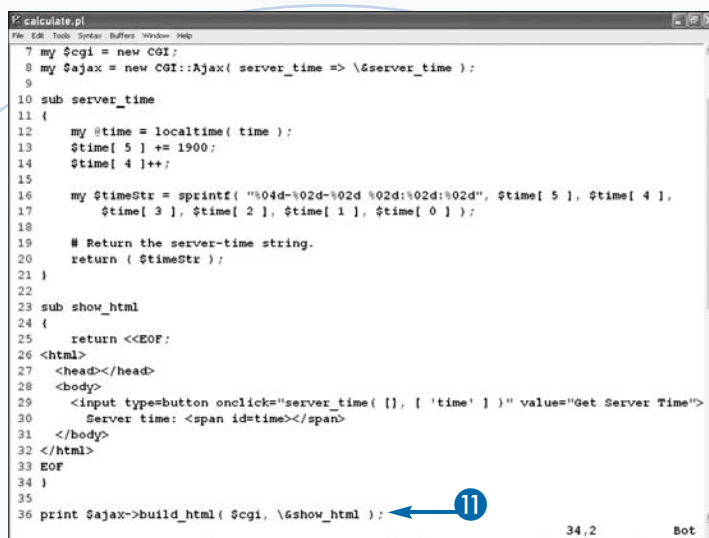
```
1 #!C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use CGI::Ajax;
6
7 my $cgi = new CGI;
8 my $ajax = new CGI::Ajax( server_time => \%server_time );
9
10 sub server_time
11 {
12     my @time = localtime( time );
13     $time[ 5 ] += 1900;
14     $time[ 4 ]++;
15
16     my $timeStr = sprintf( "%04d-%02d-%02d %02d:%02d:%02d", $time[ 5 ], $time[ 4 ],
17         $time[ 3 ], $time[ 2 ], $time[ 1 ], $time[ 0 ] );
18
19     # Return the server-time string.
20     return ( $timeStr );
21 }
22
```

```
23 sub show_html
24 {
25     return <<EOF;
26 <html>
27 <head></head>
28 <body>
29     <input type=button onclick="
30         Server time:
31     </body>
32 </html>
33 EOF
34 }
35
```

- 11 Type `print $ajax->build_html($cgi, \&show_html);`.

Note: Make sure `$cgi` is already initialized as a CGI library handle.

- 12 Save the Perl CGI script.



```

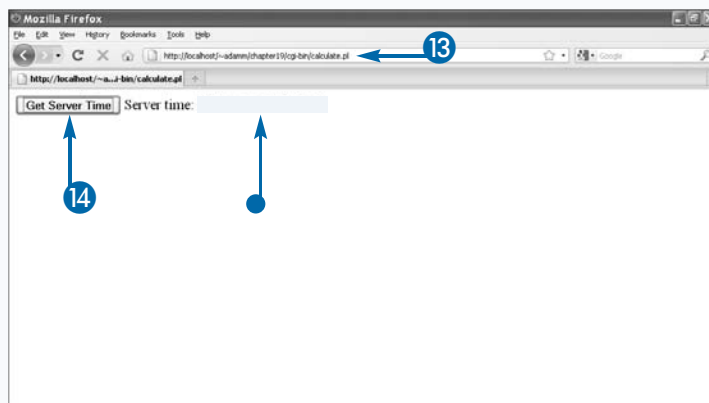
1  # calculate.pl
2
3  my $cgi = new CGI;
4
5  my $ajax = new CGI::Ajax( server_time => \&server_time );
6
7  sub server_time
8  {
9      my @time = localtime( time );
10     $time[ 5 ] += 1900;
11     $time[ 4 ]++;
12
13     my $timeStr = sprintf( "%04d-%02d-%02d %02d:%02d:%02d", $time[ 5 ], $time[ 4 ],
14                           $time[ 3 ], $time[ 2 ], $time[ 1 ], $time[ 0 ] );
15
16     # Return the server-time string.
17     return ( $timeStr );
18 }
19
20 sub show_html
21 {
22     return <<EOF;
23     <html>
24     <head></head>
25     <body>
26     <input type=button onclick="server_time( [], [ 'time' ] )" value="Get Server Time">
27     Server time: <span id=time></span>
28     </body>
29     </html>
30 EOF
31 }
32
33 print $ajax->build_html( $cgi, \&show_html );

```

- 13 Load the Perl CGI script in a Web browser.

- 14 Trigger the JavaScript function call.

- The browser displays the Perl subroutine's output.



Apply It

You can easily add `HTML::Template` into `CGI::Ajax`. Simply bypass the step where you create a `show_html` subroutine and instead place this information directly in the template. When you are ready to call `build_html`, simply reference the `HTML::Template` output method. Make sure that `build_html` and the template output methods are the last line in your code, and that you can omit `$cgi->header()`, as `build_html` takes care of that for you.

TYPE THIS

```

use HTML::Template;
my $tmpl = HTML::Template->new( filename =>
    "template.tmpl" );
# ...
print $ajax->build_html( $cgi, $tmpl->output() );

```



RESULTS

The Perl CGI script contains no actual HTML or JavaScript code; it only exists in the separate HTML template file. Regular program execution continues.

For more information on how to use `HTML::Template`, see Chapter 13.

Call Perl Subroutines Through JavaScript

You can execute Perl subroutines through your Web site's JavaScript with the CGI::Ajax module. You can use this to populate content on your Web site with data that must originate from your Web server, without the need to reload the entire Web page.

```
my $ajax = CGI::Ajax->new( jsFunction =>
    \&perl_subroutine );
```

If you have additional Perl subroutines that you want to execute through JavaScript, you can either append to this list when you initialize the CGI::Ajax object, or use the `register` method. Once you have registered your Perl subroutine in CGI::Ajax, you can define it in your script as a normal subroutine:

```
sub perl_subroutine {
    my ( $input1, $input2, ... ) = @_;
    # ...
}
```

Call Perl Subroutines Through JavaScript

- 1 Open a Perl CGI script in a text editor that imports CGI::Ajax.
- 2 Type `$ajax->register(jsFunction => \&perl_subroutine);`.

Note: You can call `register` if you already have one function registered during `new`. Alternatively, you can register multiple functions during `new` simultaneously.

- 3 Create a `perl_subroutine` subroutine.
- 4 Type `my (input) = @_;` to accept variables as input.
- 5 Use the subroutine's input data to produce output data.
- 6 Type `return (output);` to return the output as an array.

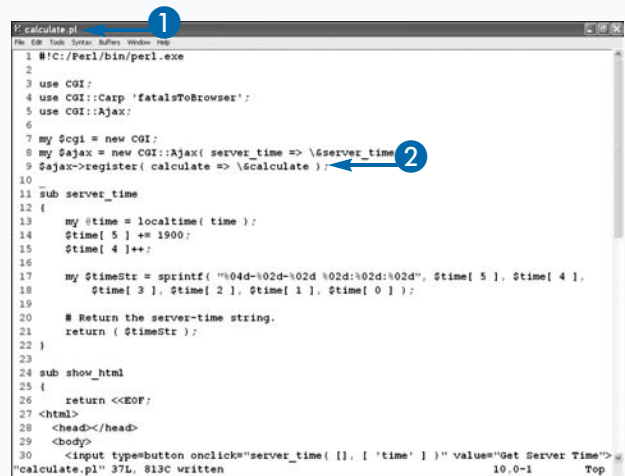
```
return $output;
```

```
}
```

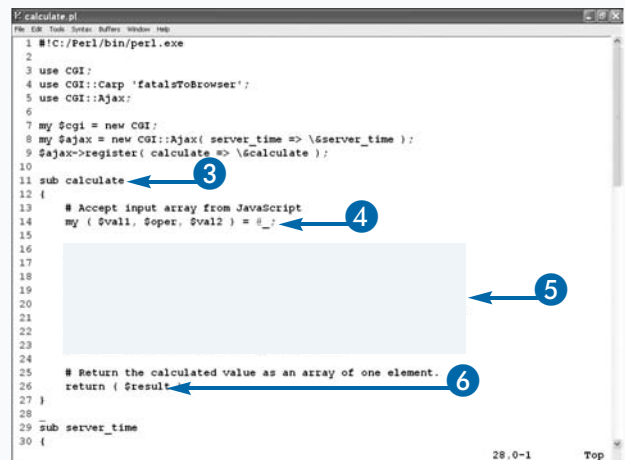
To call the function with JavaScript, the syntax is fairly specific. Two parameters are accepted: the first one represents the values you want to pass to Perl; the second parameter indicates what CSS ID in your HTML will display the Perl output. Both arguments must be represented as an array, hence the square brackets:

```
jsFunction( ['var1','var2',... ], ['result']
    );
```

When you call your function in JavaScript, CGI::Ajax assumes you want to output the Perl result into the HTML output of the site. You must ensure in your HTML code that you have something that references CSS ID used as the second parameter of the JavaScript function. Placing `` anywhere is sufficient.



```
1 #!/C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use CGI::Ajax;
6
7 my $cgi = new CGI;
8 my $ajax = new CGI::Ajax( server_time => \&server_time );
9 $ajax->register( calculate => \&calculate );
10
11 sub server_time
12 {
13     my $time = localtime( time );
14     $time[ 5 ] += 1900;
15     $time[ 4 ]++;
16
17     my $timeStr = sprintf( "%04d-%02d-%02d %02d:%02d:%02d", $time[ 5 ], $time[ 4 ],
18         $time[ 3 ], $time[ 2 ], $time[ 1 ], $time[ 0 ] );
19
20     # Return the server-time string.
21     return ( $timeStr );
22 }
23
24 sub show_html
25 {
26     return <<EOF;
27 <html>
28 <head></head>
29 <body>
30 <input type=button onclick=server_time( [ ], [ 'time' ] )" value="Get Server Time">
"calculate.pl" 37L, 813C written 10,0-1 Top
```



```
1 #!/C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use CGI::Ajax;
6
7 my $cgi = new CGI;
8 my $ajax = new CGI::Ajax( server_time => \&server_time );
9 $ajax->register( calculate => \&calculate );
10
11 sub calculate
12 {
13     # Accept input array from JavaScript
14     my ( $val1, $opex, $val2 ) = @_;
15
16     # ...
17
18     # Return the calculated value as an array of one element.
19     return ( $result );
20 }
21
22 sub server_time
23 {
24
25
26
27 }
28
29 sub server_time
30 {
```


- 7 Scroll to the `show_html` subroutine.
- 8 Type `jsFunction(['input'], ['display'])` to call the Perl subroutine `perl_subroutine`.

Note: In this example, you have three input fields that you want to trigger your registered Perl function.

- 9 Type ``.

Note: You can use any HTML tag as the display output, as long as you reference the CSS ID value as display output only once.

- 10 Save the Perl CGI script.

- 11 Open the Perl CGI in a Web browser.

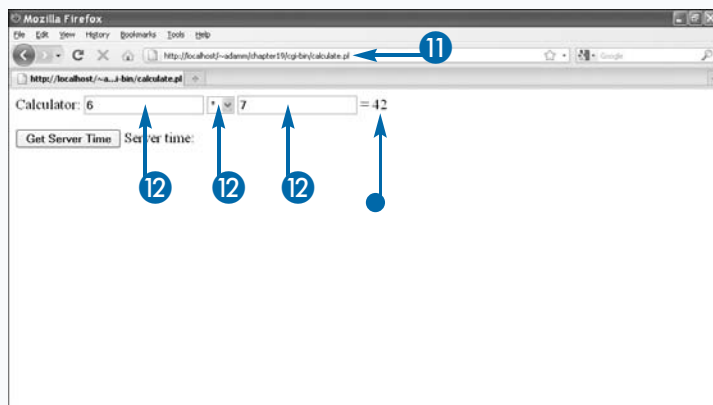
- 12 Trigger the JavaScript function call.

- The browser displays the Perl subroutine's output.

```

1  calculate.pl
2  #!/usr/bin/perl
3  use strict;
4  use warnings;
5  use CGI;
6  my $cgi = CGI->new;
7  return ( $timeStr );
8  }
9  sub show_html
10 {
11     return <<EOF>>;
12     <html>
13     <head></head>
14     <body>
15     Calculator:
16     <input type=text id=val1
17     onkeyup="
18     <select id=oper onchange="
19     <option value="+>+
20     <option value="->-
21     <option value="*>*
22     <option value="/>/
23     </select>
24     <input type=text id=val2
25     onkeyup="
26     =
27     <p>
28     <input type=button onclick="server_time( [], [ 'time' ] )" value="Get Server Time">
29     Server time: <span id=time></span>
30     </body>
31     </html>
32     EOF
33 }
34 print $cgi->build_html( $cgi, \%show_html );

```



Extra

Because a registered Perl function actually returns an array, you can have multiple CSS IDs represented in the second parameter of the JavaScript function. This allows a single Perl subroutine to produce output in multiple locations of your HTML code during a single function call.

For example, if the user types in any non-numeric characters, Perl will fail to evaluate the equation. It would be useful to display the Perl error message to the user, so add the following code:

```
calculate( [ 'val1', 'oper', 'val2' ], [ 'result', 'errmsg' ] );
```

Later in the HTML file, display the error message by referencing the second output CSS ID called `errmsg`:

```
Perl evaluation error message, if any: <span id=errmsg></span>
```

In the `calculate` subroutine, modify the `return` line to return the error message from the preceding `eval` function as the second element of the array output. The error message is automatically accessible from the special variable `$@`:

```
return ( $result, $@ );
```

Reload the page. Now if you enter in only numbers, the browser does not display an error message; however, if you enter any other characters, the Perl error message appears dynamically in the HTML.

Call JavaScript Through Perl Subroutines

After CGI::Ajax calls a registered Perl subroutine, it is possible to forward that subroutine's output to another JavaScript function, one that is outside the direct control of CGI::Ajax. This is useful if you want to further process Perl output data in JavaScript. To implement this feature, you first create a new JavaScript function, and then supply it as the only element in the second array argument of your registered JavaScript function. You must not use single quotes at all; otherwise, CGI::Ajax will treat it like a CSS ID that needs to be populated, rather than a separate function:

```
jsFunction( [ 'var1','var2',... ], [
  myFunction ] );
```

Your new JavaScript function will be able to access the output variables from Perl by way of a special arguments array, automatically declared and populated by CGI::Ajax.

Call JavaScript Through Perl Subroutines

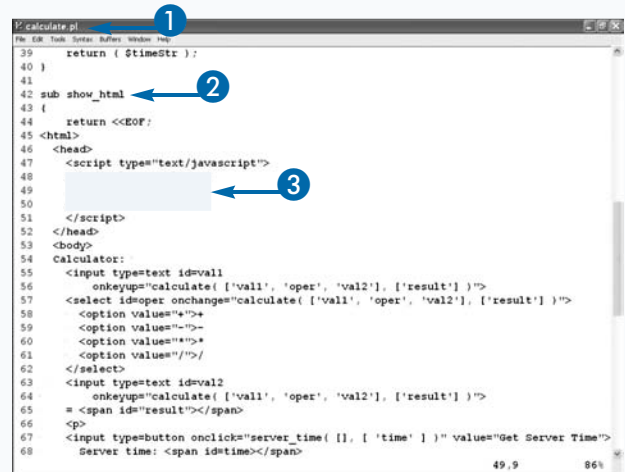
- 1 Open a Perl CGI script that uses CGI::Ajax.
- 2 Scroll to the `show_html` subroutine.
- 3 Create a JavaScript function inside your HTML output.

- 4 Type `return ($result, $@);` in the last line of the Perl subroutine that you created in the previous task.
- 5 Use a special arguments array to access the output from the Perl subroutine.
- 6 Perform a JavaScript-specific activity with the data coming from Perl.
- 7 Type `document.getElementById(display).innerHTML = variable;` to update an HTML tag with the variable's data.

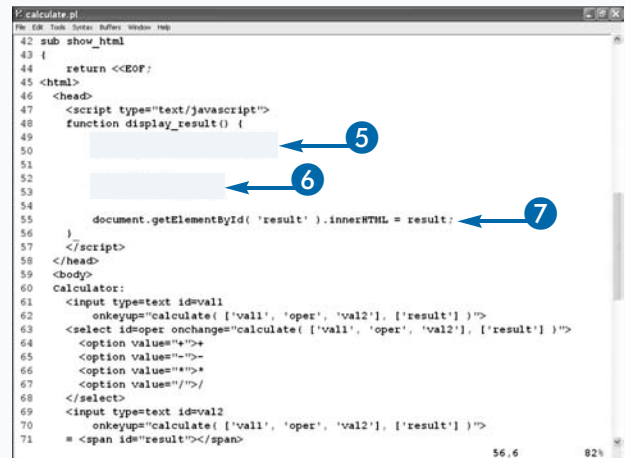
This process will disable the automatic CSS ID auto-population with your Perl subroutine's output. If you want to maintain that functionality, you need to add into your new custom function some way to get the CSS ID's object, and then assign the input to its `innerHTML` property:

```
function myFunction() {
  var perlOutput = arguments[ 0 ];
  var resultsObject =
  document.getElementById( 'result' );
  resultsObject.innerHTML = perlOutput;
}
```

If your new function does not require the ability to update any HTML tag in the Web page as program output, you can bypass the `getElementById` and `innerHTML` JavaScript commands entirely.



```
P:\calculate.pl
File Edit Tools Syntax Buffers Window Help
39 return ( $timeStr );
40 }
41
42 sub show_html ← 2
43 {
44     return <<EOF;
45 <html>
46 <head>
47     <script type="text/javascript">
48         ← 3
49
50     </script>
51 </head>
52 <body>
53 Calculator:
54 <input type="text" id=val1
55     onkeyup="calculate( ['val1', 'oper', 'val2'], ['result'] )"
56 <select id=oper onchange="calculate( ['val1', 'oper', 'val2'], ['result'] )"
57 <option value="+">+
58 <option value="-">-
59 <option value="*">*
60 <option value="/">/
61 </select>
62 <input type="text" id=val2
63     onkeyup="calculate( ['val1', 'oper', 'val2'], ['result'] )"
64 = <span id="result"></span>
65 <p>
66 <input type="button" onclick="server_time( [ [ 'time' ] ] )" value="Get Server Time">
67 Server time: <span id=time></span>
68
49.9 86%
```



```
P:\calculate.pl
File Edit Tools Syntax Buffers Window Help
42 sub show_html
43 {
44     return <<EOF;
45 <html>
46 <head>
47     <script type="text/javascript">
48         function display_result() { ← 5
49
50         }
51     </script>
52     ← 6
53     document.getElementById( 'result' ).innerHTML = result; ← 7
54 </head>
55 <body>
56 Calculator:
57 <input type="text" id=val1
58     onkeyup="calculate( ['val1', 'oper', 'val2'], ['result'] )"
59 <select id=oper onchange="calculate( ['val1', 'oper', 'val2'], ['result'] )"
60 <option value="+">+
61 <option value="-">-
62 <option value="*">*
63 <option value="/">/
64 </select>
65 <input type="text" id=val2
66     onkeyup="calculate( ['val1', 'oper', 'val2'], ['result'] )"
67 = <span id="result"></span>
68
56.6 82%
```

- 8 Type [**function**] as the second argument to all registered JavaScript functions that call Perl subroutines.
- 9 Save the Perl CGI script.

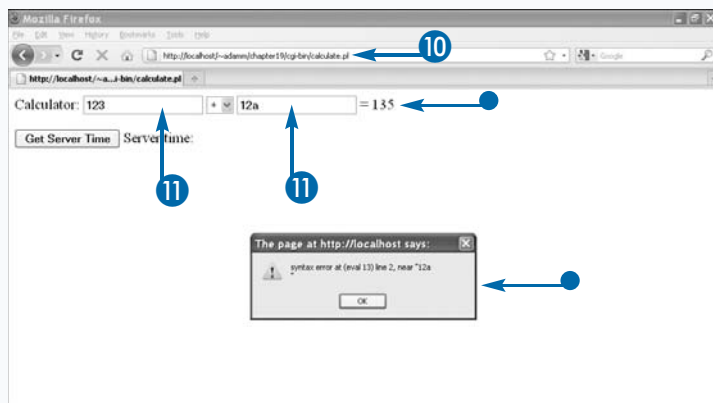
```

42 sub show_html
43 {
44     return <<EOF>;
45 <html>
46 <head>
47 <script type="text/javascript">
48 function display_result() {
49     var result = arguments[ 0 ];
50     var errmsg = arguments[ 1 ];
51
52     if ( errmsg )
53         alert( errmsg );
54
55     document.getElementById( 'result' ).innerHTML = result;
56 }
57 </script>
58 </head>
59 <body>
60 Calculator:
61 <input type="text" id=val1
62   onkeyup="calculate( ['val1', 'oper', 'val2'],
63   <select id=oper onchange="calculate( ['val1', 'oper', 'val2'],
64   <option value="+">+
65   <option value="-">-
66   <option value="*">*
67   <option value="/">/
68 </select>
69 <input type="text" id=val2
70   onkeyup="calculate( ['val1', 'oper', 'val2'],
71   = <span id="result"></span>

```

- 10 Open the Perl CGI in a Web browser.
- 11 Trigger the JavaScript function call.
 - The browser displays the Perl subroutine's output using the JavaScript call.

Note: In this example, the JavaScript alert function only displays an error message when eval produces an error.



Apply It

You can have CGI::Ajax convey new JavaScript code, generated by Perl, and have it evaluated by the Web browser as native JavaScript. This can be useful if you want to provide new, dynamic JavaScript functionality after the initial Web page load. To make this happen, use the JavaScript eval function to execute the new JavaScript code.

TYPE THIS INTO PERL

```

sub perl_subroutine {
    my ( $input ) = @_;
    return ( "alert(
        'Received input: $input' )" );
}

```

TYPE THIS INTO JAVASCRIPT

```

function myFunction() {
    var perlJsCode = arguments[ 0 ];
    eval( perlJsCode );
}

```

RESULTS

When CGI::Ajax calls your registered Perl subroutine, Perl returns JavaScript code. When the corresponding output function is called by CGI::Ajax, it uses eval to execute the custom JavaScript code as if it were written in-line.

Enable Debug Mode in CGI::Ajax

You can enable a debug mode in CGI::Ajax to help you diagnose any communication problems between Perl and JavaScript. Because AJAX interactions occur behind the regular Web browser's display output, it can be harder to identify if a JavaScript function is working incorrectly, or if Perl is responding to the request inappropriately. Prior to calling `build_html`, CGI::Ajax provides two methods that allow you to see the activity that is going on behind its interface:

```
$ajax->JSDEBUG( number );
```

If you set `JSDEBUG`'s number to 1, CGI::Ajax prints the internal URL it is using to call your Perl CGI script. The Web site's JavaScript source code for your registered functions is also visible. If you set `JSDEBUG`'s number to 2, the internal URL is still displayed, but the complete CGI::Ajax JavaScript code is now visible.

Enable Debug Mode in CGI::Ajax

- 1 Open a Perl CGI script that uses CGI::Ajax.
- 2 Scroll to the `build_html` method.
- 3 Type `$ajax->JSDEBUG(2);` to enable full JavaScript debugging without any compression.
- 4 Type `$ajax->DEBUG(1);` to show debug information in the Apache logs.
- 5 Save the CGI script.

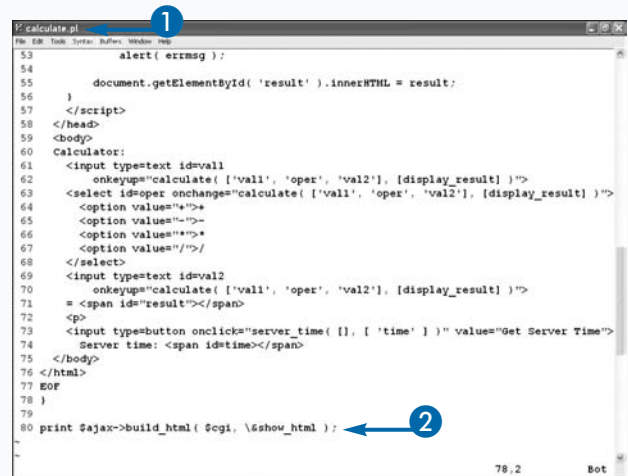
Clicking that URL in your Web browser allows you to view the raw output being produced by your Perl CGI script's registered subroutine. This is useful if you believe that there is a problem in your Perl CGI script:

```
$ajax->DEBUG( number );
```

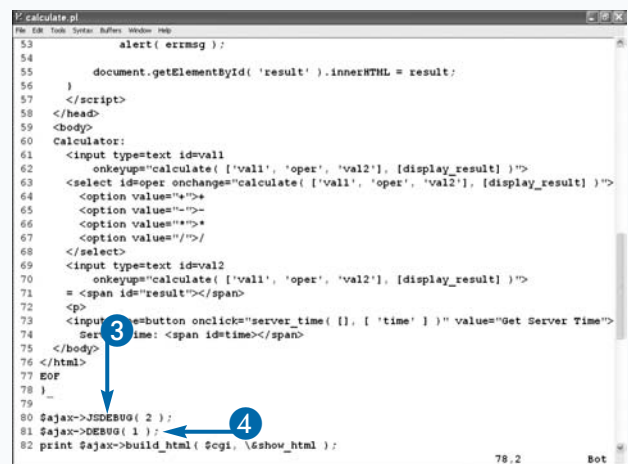
If you set this debug parameter to 1, additional data concerning the Perl functionality of CGI::Ajax is printed to the Apache error log. The default value for both is 0.

The log directory Apache stores its log files into differs, depending on your operating system. On Windows, this is `Apache Install Dir/logs`, on Debian and Ubuntu this is `/var/log/apache2/`, and on Red Hat this is `/var/log/httpd/`. In your Perl code, you can always insert arbitrary debug information into the Apache logs by printing to standard-error (STDERR):

```
print STDERR "Log Message";
```



The screenshot shows a Perl CGI script named `calculate.pl` in a text editor. The script contains HTML and JavaScript code for a calculator. A blue circle with the number 1 is placed above the `print $ajax->build_html($cgi, \%show_html);` line. A blue arrow with the number 2 points to this line.



The screenshot shows the same Perl CGI script `calculate.pl` with two additional lines of code added at the bottom, indicated by blue circles with numbers 3 and 4. Line 80 is `$ajax->JSDEBUG(2);` and line 81 is `$ajax->DEBUG(1);`. A blue arrow with the number 3 points to line 80, and a blue arrow with the number 4 points to line 81.

- 6 Open the Perl CGI script in a Web page.
- 7 Interact with the Web site to trigger AJAX functionality.

- The browser displays CGI::Ajax's internal URL.

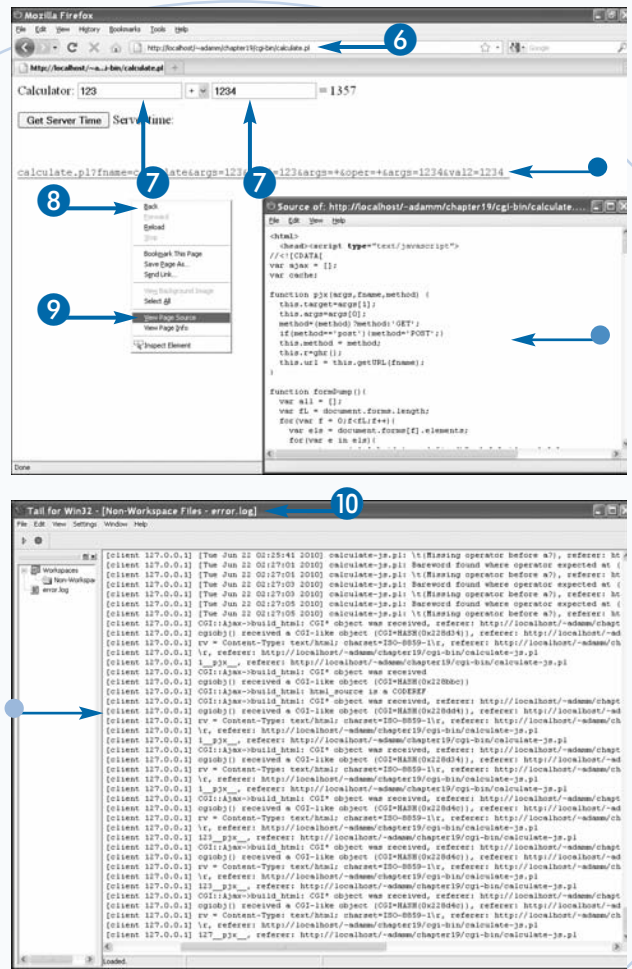
- 8 Right-click the Web page.

- 9 Click View Page Source.

- The uncompressed JavaScript produced by CGI::Ajax.

- 10 View the Apache Error Log.

- Activity produced by CGI::Ajax in Perl.



Extra

Using a *profiler* in your Web browser is another way to gain a new perspective on the low-level communication between your browser and the Web server. A profiler helps you identify what requests are generated, how they are responded to, and how the browser interprets the response on the Web page. It also allows you to investigate and debug core JavaScript syntax errors and other problems.

If you use Google Chrome, you can access a built-in profiler by pressing Ctrl+Shift+I. Other available features include an inspector, script reviewer, timeline summary, local storage, and auditing for network utilization and Web page performance.

If you use Mozilla Firefox, Firebug is an excellent add-on program that contains a profiler, inspector, logger, analyzer, and layout examiner, all installed using a single extension. In Firefox, click Tools → Add-ons → Browse All Add-ons to locate Firebug.

If you use Microsoft Internet Explorer, you can install the Internet Explorer Developer Toolbar as a free add-on. Search the Microsoft Developers Network, or MSDN, for a download link for your region.

Integrate Perl and XML

You can use Perl to interact with XML-formatted communications between third-party services, or within your own JavaScript AJAX code, using the third-party Perl module named `XML::Simple`. This module is not a part of the standard Perl distribution and you must install it manually. See Chapter 9 for more information about installing third-party modules.

XML is similar to HTML, except its tag names are entirely open. While it theoretically has the same structure as a Perl hash reference, in practice XML does support features that do not translate well into Perl. `XML::Simple` is one parser implementation that makes several assumptions on your behalf so that you can simplify an XML tree into a Perl hash reference. When you receive XML text data, you can convert it into a hash reference with `XMLin`:

```
use XML::Simple;
$hashref = XMLin( $xmlText );
```

Likewise, if Perl has its own hash reference of data that you want to send to convert into XML, you can use `XMLout`:

```
$hashref = { key1 => arrayref, key2 =>
    hashref, key3 => scalar, ... };
$xmlText = XMLout( $hashref );
```

Parsing XML in Perl, or in any language, is not a simple process. You may find that `XML::Simple` makes too many assumptions and so you require a more fine-grained parser. CPAN contains dozens of XML parsers. Fortunately, there is a Perl-XML FAQ that summarizes the strengths and weaknesses of the more popular parsers, allowing you to make an informed decision. You can find it at <http://perl-xml.sourceforge.net/faq/>.

Integrate Perl and XML

- 1 Type `use XML::Simple;`
- 2 Read an XML stream or text file into Perl.
- 3 Examine the original XML input data.
- 4 Type `my $hashref = XMLin($xml);`.
- 5 Examine the output of the `XML::Simple` hash reference with `Data::Dumper`.
- 6 Type `$xml = XMLout($hashref);`.
- 7 Examine the output of the `XML::Simple` conversion.
- 8 Run the program to view the original XML data, the converted hash reference, and the re-converted XML data.

Note: If you lose any data between the original and the two conversion steps, `XML::Simple` may not be appropriate for you. Read the Perl-XML FAQ for more information.

```
1 #!C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use CGI::Ajax;
6 use XML::Simple;
7 use Data::Dumper;
8
9 my $cgi = new CGI;
10 my $ajax = new CGI::Ajax( server_time => \&server_time );
11 $ajax->register( calculate => \&calculate );
12 $ajax->register( read_xml => \&read_xml );
13
14 sub get_xml
15 {
16     my ( $xml ) = @_;
17
18     print STDERR "Original XML: " . $xml;
19
20     my $xmlHashRef = XMLin( $xml );
21     print STDERR "Converted XML: " . Dumper( $xml );
22 }
23
24
25 sub calculate
26 {
27     # Accept input array from JavaScript
28     my ( $val1, $opex, $val2 ) = @_;
29
30     # If either value is blank, assume it to be a zero.
```

```
22
23
24     $xml = XMLout( $xmlHashRef );
25     print STDERR "Reconstructed XML: " . $xml;
26 }
27
28 sub calculate
29 {
30     # Accept input array from JavaScript
31     my ( $val1, $opex, $val2 ) = @_;
```


Integrate Perl and JSON

You can use Perl to interact with JSON-formatted communications between third-party services, or within your own JavaScript AJAX code, using the third-party Perl module simply named JSON. This module is not a part of the standard Perl distribution and you must install it manually. See Chapter 9 for more information about installing third-party modules.

Because JSON is merely JavaScript code written in a structured object notation, the idea is actually very close to a Perl hash reference. Perl can navigate through a hash reference with ease, and JavaScript can do the same for JSON, so all you need is a way to convert data from one format into the other.

When you receive a JSON text file, you can convert it into a hash reference with `from_json`:

```
use JSON;
$hashref = from_json( $jsonText );

Likewise, if Perl has its own hash reference of data that
you want to send, to convert to JSON you can use
to_json:

$hashref = { key1 => arrayref, key2 =>
    hashref,
    key3 => scalar, ... };
$jsonText = to_json( $hashref );
```

You can now take this data and return it to CGI::Ajax for processing within a JavaScript function. For more information about JSON, see the official JSON Web site at www.json.org/.

Integrate Perl and JSON

- 1 Type `use JSON`;
- 2 Read a JSON stream or text file into Perl.
- 3 Examine the original JSON input data.
- 4 Type `my $hashref = from_json($json);`
- 5 Examine the output of the JSON hash reference with `Data::Dumper`.
- 6 Type `$json = to_json($hashref);`.
- 7 Examine the output of the JSON conversion.
- 8 Run the program to view the original JSON data, the converted hash reference, and the re-converted JSON data.

```
1 #!C:/Perl/bin/perl.exe
2
3 use CGI;
4 use CGI::Carp 'fatalToBrowser';
5 use CGI::Ajax;
6 use JSON;
7 use Data::Dumper;
8
9 my $cgi = new CGI;
10 my $ajax = new CGI::Ajax( server_time => \&server_time );
11 $ajax->register( calculate => \&calculate );
12 $ajax->register( read_JSON => \&read_JSON );
13
14 sub get_JSON
15 {
16     my ( $json ) = @_;
17
18     print STDERR "Original JSON: " . $json;
19
20     my $jsonHashRef = from_json( $json );
21     print STDERR "Converted JSON: " . Dumper( $jsonHashRef );
22 }
23
24
25 sub calculate
26 {
27     # Accept input array from JavaScript
28     my ( $val1, $oper, $val2 ) = @_;
29
30     # If either value is blank, assume it to be a zero.
```

```
23
24     $json = to_json( $jsonHashRef );
25     print STDERR "Reconstructed JSON: " . $json;
26 }
27
28 sub calculate
29 {
30     # Accept input array from JavaScript
31     my ( $val1, $oper, $val2 ) = @_;
```

Introducing PayPal

If you have a product or service to sell, it makes sense to provide it for sale on the Internet. In the early days of the Internet, many startup companies appeared, promising secure credit card transactions and instant payment transfers to serve a growing digital economy. Since that time, some companies have disappeared but one that has remained viable is PayPal.

PayPal offers multiple solutions and products for both consumers and merchants. You can use PayPal as a financial transaction engine for your Web site, charging credit cards, issuing refunds, transferring funds, and as a database for organizing your orders, transactions, and shipping addresses.

PayPal Integration Options

PayPal provides several interfaces for online merchants to process credit card transactions. Each solution is tailored to the size of your Web site and the amount of e-commerce traffic you can expect to process.

PayPal Encrypted Web Payments

The Encrypted Web Payments (EWP) API is a programming interface that is easiest to set up on Web sites where you want to implement credit card payment transactions. After you sign up for the EWP service, PayPal helps you create a payment button that you can place anywhere on your Web site. When users click the button, they are redirected to the PayPal Web site where their payment is processed.

Ideally, you should already have a catalog of products or services available on your Web site that users can easily browse. Adding the PayPal EWP button allows your users to purchase the item now, or to add the item to a PayPal-managed shopping cart.

This API requires no back-end Perl CGI integration. You can use the PayPal setup program to construct any PayPal buttons you require. The program provides you with static HTML code that you can simply copy and paste into your Web site.

PayPal API

The PayPal API is a service that allows you to process PayPal transactions directly from your Web site's Perl CGI scripts. This is a hands-on approach to credit card transactions, meaning you are in charge of producing your own online catalog, shopping cart, and checkout application, and managing your Web site's security and private user information. The end-user never knows that PayPal is being used as a financial transaction engine for their online purchase on your Web site.

The PayPal API provides several integration services. The most common service used by Web sites is the Express Checkout. This allows you to redirect the user to the PayPal Web site for payment processing after they have amassed items in a shopping cart. You can also issue a one-time credit card transaction directly with Express Checkout. Both of these services are demonstrated in this chapter.

Other services, such as Website Payments Pro, Mobile Checkout, and Mass Payment, are also available. For more information about all the API services that are available, see the PayPal Tools for Developers Web site at <https://www.x.com/community/ppx/dev-tools>.

NVP Interface

To call a PayPal API service from your Web site, you can use the Name-Value Pairs, or NVP, interface. This is the recommended method to call the PayPal API, which utilizes simple HTTP POST transactions that follow a predetermined CGI parameter structure.

The remainder of this chapter uses the NVP interface to demonstrate how to integrate the PayPal API into your Web site. You can learn more about using NVP at the PayPal developers' Web site at <https://www.x.com/docs/DOC-1242>.

SOAP Interface

The Simple Object Access Protocol, or SOAP, is an alternative method you can use to call the PayPal API. SOAP is an XML-based communication protocol that uses WSDL and XSD files to define the PayPal API function structure. If you are already familiar with SOAP in general, you can use the Perl SOAP::WSDL module to interface with the PayPal API.

PayPal Integration Options *(continued)***PayPal Instant Payment Notification**

The Instant Payment Notification (IPN) API is a PayPal service that sends a notification to your Web site when a payment transaction has been completed. This service is intended for larger e-commerce Web sites that deal with hundreds of transactions per day.

If you choose not to implement IPN, you will still be notified of transactions by way of regular e-mail; however, this means that you require a person to confirm that a payment has been received before an order can be fulfilled. IPN automates this process by allowing you to link your back-end fulfillment software with the PayPal transaction engine.

PayPal Environments

PayPal offers two different environments for you to use to familiarize yourself with its services.

Sandbox Account

The PayPal sandbox is a special environment that is an exact clone of the live system. You can use this environment to test your Web site, the PayPal APIs, and your Perl scripts using the APIs, and review the e-mail and GUI experienced by all parties.

When you sign up for a sandbox account, you have the option to create as many dummy buyer or seller accounts as you need. This way, you can post fictional transactions and be able to experience exactly what the customer, merchant, and PayPal experience through every stage of the process. The sandbox account is free, has no fees associated with it, and you can sign up without being subject to any approval process.

Live Account

When you are ready to perform real transactions, you can sign up for a live PayPal account. Depending on the types of transactions you are performing, there is a set registration and approval process dictated by PayPal. For more information, go to <https://merchant.paypal.com>.

PayPal Perl Modules

You can install third-party Perl modules that interface with various PayPal services. These modules were developed by other Perl developers and not PayPal, and were derived from public API documentation for other officially supported languages.

Business::PayPal::NVP

You can use the Perl module `Business::PayPal::NVP` to execute any method call that PayPal has made available on its NVP API. You can use this module to access the following services: Address Verify, Direct Payment, Express Checkout, Get Balance, Get Transaction Details, Mass Payment, Recurring Payments, Refund Transaction, and Transaction Search.

Because the NVP API is constantly changing, you need to refer to the official NVP documentation for a current list of available methods and arguments. You can find this information at https://cms.paypal.com/cms_content/US/en_US/files/developer/PP_NVPAPI_DeveloperGuide.pdf.

Business::PayPal::IPN

The Perl module `Business::PayPal::IPN` allows you to confirm that online transactions are valid and legitimate after a monetary transaction has been completed on the PayPal network. By writing a Perl script to independently check a transaction's status, you ensure that your Web site cannot be duped into fulfilling an order that has not yet been completed.

Sign Up for a PayPal Sandbox Account

You can get access to the PayPal sandbox environment by signing up for a sandbox account on the PayPal Developers' Web site. The sandbox environment is an exact clone of the same PayPal APIs you will find in the live environment, except that all transactions posted by you will not involve any real money. This is a very useful way to validate that your Perl CGI scripts are working correctly, without risking any *costly* mistakes.

PayPal recommends that all new developers test their applications in the sandbox environment prior to deployment. This way you can ensure that it functions as you intend and within the guidelines set forth by PayPal.

You can sign up for a sandbox account by going to the PayPal Developers' Web site at <https://developer.paypal.com>.

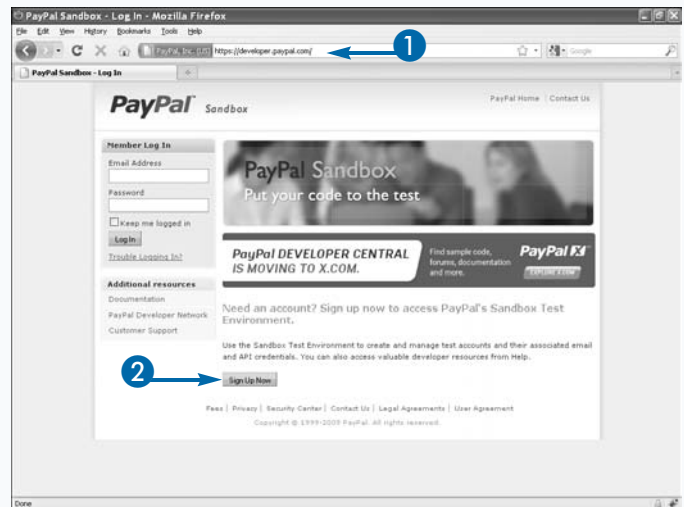
Once you sign up, you can create fictional buyer and seller accounts, and even access the e-mail inbox of those accounts. You will be able to link these accounts into your Perl CGI scripts later and simulate transactions, query your transaction log, query your PayPal balance, and debit and credit other fictional accounts.

Your Perl CGI scripts require you to enter in API credentials in order to facilitate NVP and IPN API communications. Here you can create these credentials and plug them directly into your Perl scripts.

Be sure to review the PayPal Sandbox User Guide document for additional information. You can download a PDF from the Web site at https://cms.paypal.com/cms_content/US/en_US/files/developer/PP_Sandbox_UserGuide.pdf.

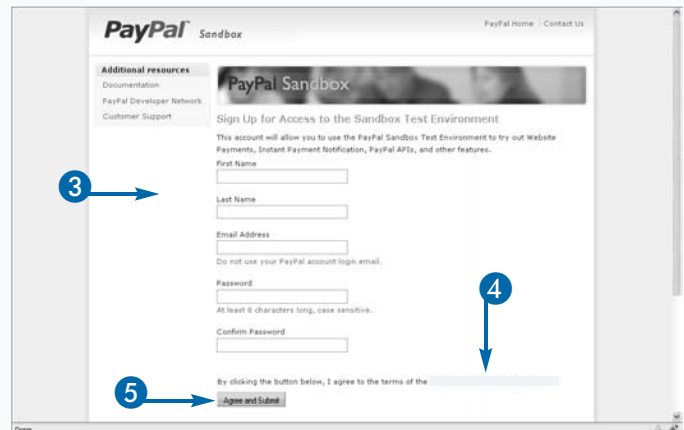
Sign Up for a PayPal Sandbox Account

- 1 Open a Web browser and go to <https://developer.paypal.com>.
- 2 Click Sign Up Now.



The Sandbox Signup form appears.

- 3 Enter your personal information to create an account.
- 4 Click here to review the PayPal Sandbox User Agreement document.
- 5 Click Agree and Submit.

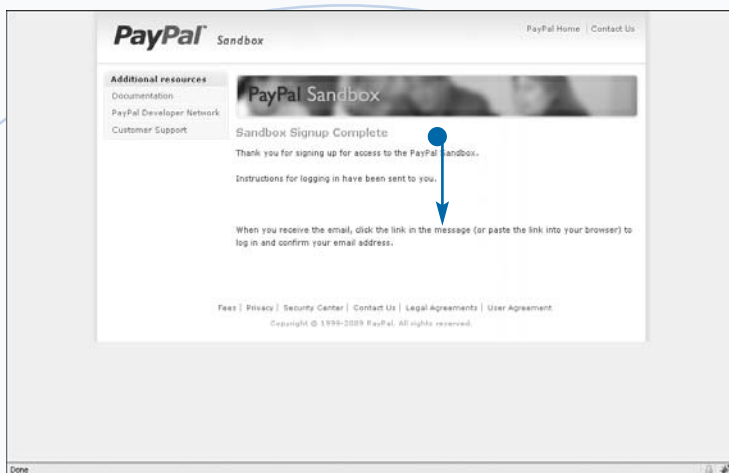


The Sandbox Signup Complete screen appears.

- An e-mail message is sent to your address.

6 Open your e-mail client.

7 Click the link in your e-mail to activate your developer account.



The login screen appears.

- Your developer account is now created and activated.



Extra

You can now enter your credentials on the front page of the PayPal Developer Sandbox Web site and log in. You can access the PayPal Sandbox administrator and set up your fictional users to represent buyers and sellers in your sandbox environment.

Remember that all activity in the sandbox is segregated, but only from the live PayPal network. You can theoretically interact with other sandbox users and developers if you know their PayPal Sandbox e-mail address. To prevent developers from accidentally creating duplicate fictional accounts, you are assigned a convoluted e-mail address:

```
username_timestamp_accounttype@mydomain.com
```

Note that you still access the sandbox by authenticating with a real PayPal Developers' Network login ID that you just created.

Once you are satisfied with how your Perl CGI scripts react to the PayPal APIs, it is time to create a real account for yourself. This setup process requires you to actually own a business account profile on PayPal, which does require a complex validation setup procedure. To sign up, go to the PayPal Merchant Services Web site at <https://merchant.paypal.com>.

Create Buyer and Seller Sandbox Accounts

You must create test accounts to represent a fictional buyer and seller in your PayPal sandbox. With these accounts you can represent yourself as a fictional customer or merchant using your Web site. If you are just starting out with PayPal, you should create test accounts using the preconfigured method, as opposed to creating one manually. This allows you to create fictional buyers and sellers quickly. If you want to do post-credit card transactions, as described later in this chapter, be sure to set up your seller account as either a Website Payments Standard or Website Payments Pro account. This way, even your fictional transactions will succeed as the sandbox environment ensures that your business account, even the fake one, has the proper settings in order to allow these types of transactions.

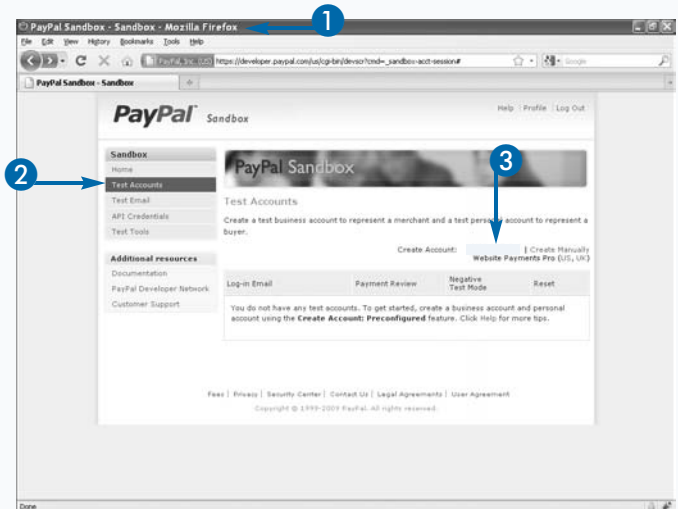
Extra

When you create your buyer and seller sandbox accounts, use a logical login name. For example, use *user1* for the buyer account, and *store1* for the seller account. When you are finished, you should see two test accounts on the Test Accounts summary page.

You can now click the Enter Sandbox Test Site link and authenticate into a false PayPal Web site using either your buyer or seller test account.

Create Buyer and Seller Sandbox Accounts

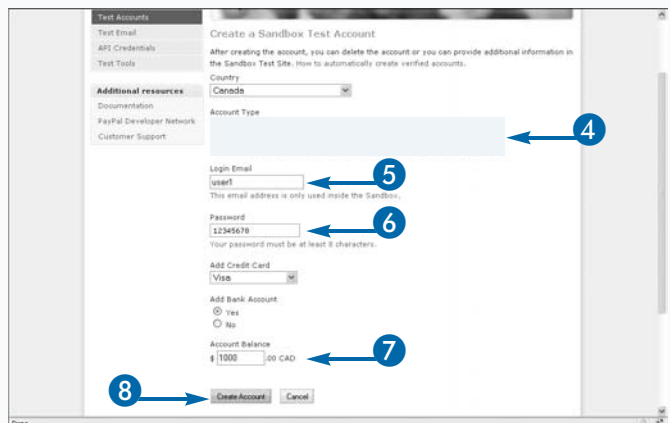
- 1 Log on to your PayPal Developer Sandbox account.
- 2 Click Test Accounts.
The PayPal Sandbox Test Accounts summary screen appears.
- 3 Click Preconfigured to create a preconfigured test account.



The Create a Sandbox Test Account screen appears.

- 4 Select the account type.
- 5 Assign a login name for the account.
- 6 Assign a password.
- 7 Assign an account balance.
- 8 Click Create Account.

The fictional account is created in your PayPal sandbox.



Retrieve Your Seller's Sandbox API Credentials

In order to use your sandbox seller account in your Perl CGI scripts, you must retrieve the account's API credentials from the PayPal Developers' Sandbox Web site. The API credentials consist of three fields: your API username, API password, and signature.

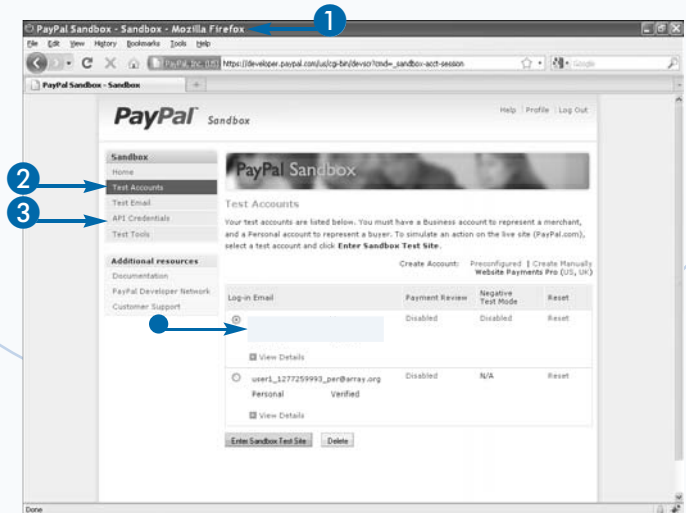
Your API username is derived from your test account username. Your API password is automatically generated based upon a timestamp, and is not related to your test account password. Your signature is also automatically generated, but using a form of private-key encryption. In other words, the signature is unique to this account, and is generated by combining your API username and API password with a private key owned by PayPal.

PayPal recommends that you keep all three values private, especially the password and signature. This is particularly important when using an API in the live environment. The onus is on you as the Web developer to properly secure this information; otherwise, anyone could authenticate as your merchant account on the PayPal API.

All three fields must exactly match this Web site; otherwise, you will not be able to authenticate into your account. When you are ready to have a real merchant account on PayPal, you will be given live versions of these fields, which you can configure into your Perl CGI scripts. However, the process of accessing your account's signature, as shown here, is slightly different between the two environments. In the sandbox, it is automatically generated for you. In the live environment, you must generate your own private key according to the procedure described on the PayPal Merchant Web site.

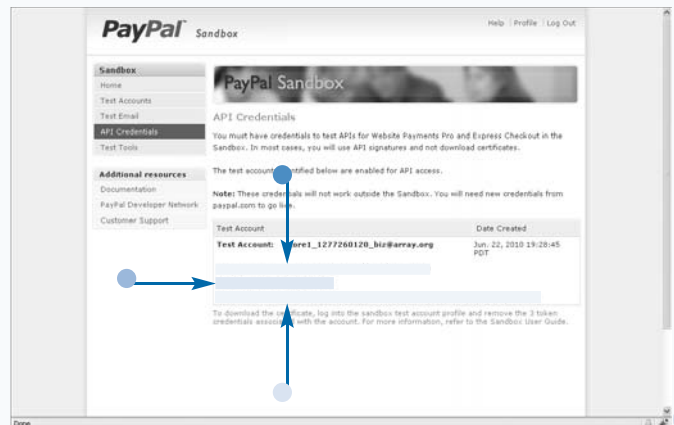
Retrieve Your Seller's Sandbox API Credentials

- 1 Log on to your PayPal Developer Sandbox account.
- 2 Click Test Accounts.
 - You need to have a seller test account created.
- 3 Click API Credentials.



The API Credentials page appears.

- Your test account API username.
- Your API password.
- Your signature.



Use Business::PayPal::NVP to Connect to PayPal

You can use the third-party Perl module `Business::PayPal::NVP` to connect to your seller PayPal account, provided that you have the account properly set up. You must have the `Business::PayPal::NVP` module installed on your workstation or Web server in order to connect to PayPal. You can find instructions for doing this in Chapter 9.

To connect, you need to provide your username, password, and signature that you retrieved from either your PayPal seller sandbox account, or your live PayPal merchant account:

```
my $pp = Business::PayPal::NVP->new( test => {  
    user => username, pwd => password, sig =>  
    signature,  
    version => num } );
```

You can optionally specify the API version number when initializing the `Business::PayPal::NVP` module. This value is sent to the PayPal NVP servers when requesting methods and can be used specify which API methods are available, according to the API documentation you are currently referencing. As of August 2010, the latest API version available is 63.0. This task demonstrates how to connect Perl to your seller's sandbox account. If you are connecting to a live account, the Apply It section gives you information about this.

You can get a complete list of the available API methods by downloading the NVP API document from the PayPal Developer Network at <https://www.x.com/community/ppx/documentation>. On this Web page, click Express Checkout, and select the PDF download for the NVP API Developer Guide and Reference. You can find the latest API version available by looking at the document's revision history.

Use Business::PayPal::NVP to Connect to Paypal

- 1 Open a new Perl script in a text editor.
- 2 Type `use Business::PayPal::NVP;`
- 3 Type `my $pp = Business::PayPal::NVP->new`
`(test => {`
- 4 Type `};`.
- 5 Type `user => username,`
- 6 Type `pwd => password,`
- 7 Type `sig => signature,`
- 8 Type `version => number,`

```
1 #!C:/Perl/bin/perl.exe  
2  
3 use Data::Dumper;  
4 use Business::PayPal::NVP;  
5  
6 my $pp = Business::PayPal::NVP->new( test => {  
7  
8 } );  
9
```

```
1 #!C:/Perl/bin/perl.exe  
2  
3 use Data::Dumper;  
4 use Business::PayPal::NVP;  
5  
6 my $pp = Business::PayPal::NVP->new( test => {  
7     user => "store1_1277260121_biz_api1.array.org",  
8     pwd => "1277260125",  
9     sig => "An44HPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LHvvg2aygppIpYB9-Dh",  
10    version => 63.0,  
11 } );  
12
```

- 9 Type `my %resp = $pp->method(arguments);` to execute an NVP method.
- 10 Type `print Dumper({ %resp });` to examine the method's response codes.
- 11 Save the Perl script.

```

paypal_connect-test.pl
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260121_biz_apil.array.org",
8     pwd  => "1277260125",
9     sig  => "An44HPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LmVvG2aygpIpyB9-Dh",
10    version => 63.0,
11    } );
12
13 my %resp = $pp->GetBalance( RETURNALLCURRENCIES => 1 );
14
15 print Dumper( { %resp } );
16

```

- 12 Open a Terminal window.
- 13 Execute the Perl script.

The Terminal window displays the response of the PayPal method.

- The PayPal method completed successfully.

```

C:\Documents and Settings\adam\My Documents\My Website\chapter20>
$VAR1 = {
    'BUILD' => '1338445',
    'L_CURRENCYCODE0' => 'USD',
    'CORRELATIONID' => '862b9270908f3',
    'VERSION' => '63',
    'ACK' => 'Success',
    'L_AMT0' => '200.00',
    'TIMESTAMP' => '2010-06-23T04:03:44Z'
};
C:\Documents and Settings\adam\My Documents\My Website\chapter20>

```

Apply It

If an error occurs, the `%resp` variable's value for the `ACK` key will hold `Failure`. If you do not use `Data::Dumper` to view the contents of `%resp`, you can at least get the error code by accessing `$resp{ 'L_ERRORCODE0' }`, and an explanation from `$resp{ 'L_LONGMESSAGE0' }`. If you still do not know why the method is failing, you can look up the error code number in the NVP API Developer Guide PDF.

When you are ready to connect your Perl code to a live PayPal merchant account, you need to make some minor changes to your module initialization code.

TYPE THIS

```

my $pp = Business::PayPal::NVP->new
( branch => live,
  live => { user => username, pwd =>
    password, sig => signature,
    version => num } );

```



RESULTS

By adding `branch => live` and changing `test =>` to `live =>`, you automatically connect to the live PayPal NVP interface. All methods you execute will now be against your live PayPal merchant account, provided your login credentials are correct for the live system.

Process a Credit Card Payment with PayPal

Once you have successfully used `Business::PayPal::NVP` to connect to your PayPal seller account, you can begin processing credit card transactions with the `DoDirectPayment` method. If you are using the PayPal sandbox, you can start doing this right away through a command-line Perl script.

At minimum, the following fields are required to use `DoDirectPayment` over the NVP API:

```
%resp = $pp->DoDirectPayment( AMT => amount,
    CURRENCYCODE => code, PAYMENTACTION =>
'Sale',
    CREDITCARDTYPE => type, ACCT => ccnumbers,
    EXPDATE => mmyyyy, IPADDRESS => ipaddr,
    FIRSTNAME => firstname, LASTNAME =>
lastname,
```

```
STREET => address, CITY => city, STATE =>
state,
    COUNTRYCODE => code, ZIP => zipcode,);
```

The `%resp` hash returns a transaction ID for the payment, and confirms the amount charged. You can use this transaction ID later for viewing the details on past payments, and for issuing refund requests. This helps you to determine for yourself exactly how the method is constructed, and the data it returns. Eventually, may want to use this method in a CGI shopping cart and checkout program of your own design, or use the PayPal Express Checkout API to handle the checkout process for you.

There are additional request and response fields that are supported by this method. Refer to Chapter 4 in the NVP API Developer Guide (`DoDirectPayment` API Operation) for a complete list of what is available and how to reference it.

Process a Credit Card Payment with PayPal

- 1 Open a Perl script that uses `Business::PayPal::NVP`.
- 2 Type `my %resp = $pp->DoDirectPayment(`.
- 3 Type `);`.

- 4 Specify the amount to charge, currency code, and payment action type for this transaction.

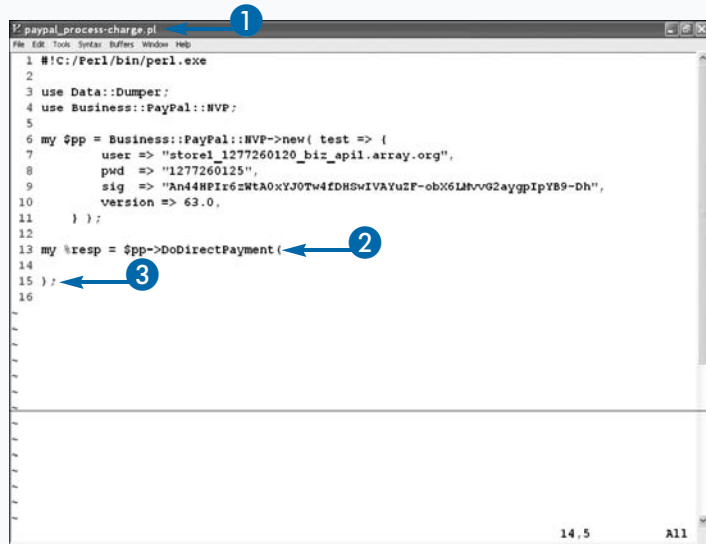
Note: The currency code must use three characters, and be a valid PayPal currency, such as `USD` or `CAD`.

- 5 Specify the credit card type, numbers, and expiry date for the transaction.

Note: Use the automatically generated VISA numbers for your sandbox buyer account.

- 6 Specify the buyer's information in the transaction.

Note: The country code must use two characters.



```
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_apil.array.org",
8     pwd  => "1277260125",
9     sig  => "An44NPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LHvvG2aygpIpYB9-Dh",
10     version => 63.0,
11 } );
12
13 my %resp = $pp->DoDirectPayment(
14
15 );
16
```



```
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_apil.array.org",
8     pwd  => "1277260125",
9     sig  => "An44NPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LHvvG2aygpIpYB9-Dh",
10     version => 63.0,
11 } );
12
13 my %resp = $pp->DoDirectPayment(
14     AMT => 14.5,
15     CURRENCYCODE => USD,
16     PAYMENTACTION => Sale,
17     CREDITCARDTYPE => VISA,
18     ACCT => 4011010101001001,
19     EXPDATE => 0101,
20     FIRSTNAME => John,
21     LASTNAME => Doe,
22     IPADDRESS => 127.0.0.1,
23     STREET => 123 Main St,
24     CITY => San Jose,
25     STATE => CA,
26     COUNTRYCODE => US,
27     ZIP => 95131,
28 );
29
30 print Dumper(%resp);
31
```

7 Use Data::Dumper to review the contents of %resp.

8 Save the Perl script.

```
paypal_process-charge.pl
File Edit Tools Syntax Buffers Window Help
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_apil.array.org",
8     pwd  => "1277260125",
9     sig  => "An44HFir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LMvVG2aygpIPYB9-Dh",
10    version => 63.0,
11  });
12
13 my %resp = $pp->DoDirectPayment(
14     AMT => '129.95', CURRENCYCODE => 'CAD', PAYMENTACTION => 'Sale',
15     CREDITCARDTYPE => 'VISA', ACCT => '4588816770456185', EXPDATE => '062015',
16     IPADDRESS => '192.168.0.1', FIRSTNAME => 'Mister', LASTNAME => 'Account-Holder',
17     STREET => '123 Fake St.', CITY => 'Springfield', STATE => 'OH',
18     'COUNTRYCODE' => 'US', ZIP => '12345'
19 );
20
21 print Dumper( { %resp } ); ← 7
22
```

9 Open a Terminal window.

10 Execute the Perl script.

The credit card payment is processed.

- The response hash of the request.
- The transaction ID of the request.

```
Command Prompt
C:\Documents and Settings\adam\My Documents\My Website\chapter20>
$VAR1 = {
    'CURRENCYCODE' => 'CAD',
    'CVV2MATCH' => 'M',
    'AMT' => '129.95',
    'TIMESTAMP' => '2010-06-23T05:55:20Z',
    'AVSCODE' => 'X',
    'BUILD' => '1338445',
    'TRANSACTIONID' =>
    'CORRELATIONID' => '1c3235c9e6c43',
    'VERSION' => '63',
    'ACK' => 'Success'
};
C:\Documents and Settings\adam\My Documents\My Website\chapter20>
```

Extra

PayPal allows you to record additional value amounts related to your transaction. Only the final amount, `AMT`, is required but you may also configure a value for `SHIPPINGAMT`, `HANDLINGAMT`, and `TAXAMT`. All monetary values must be less than \$10,000.00, and be represented with two decimal places.

It is even possible for you to summarize into your PayPal transaction the *shopping cart* items that the buyer is purchasing. This will allow you to store the information in PayPal so that both the buyer and seller can review exactly what was sold in this transaction. These fields include `L_NAMEn` for the item name, `L_DESCn` for the item description, `L_AMTn` for the cost, and `L_TAXAMTn` for the sales tax. Each field ends in a number, representing the item number being purchased. This number always starts counting at zero.

Note that the `IPADDRESS` field is used for fraud protection. If you are using this method on a Perl CGI Web site, use the variable `$ENV{ 'REMOTE_ADDR' }`. If you are just testing this method in a sandbox environment, enter any value that is formatted like an IP address.

Use the PayPal Express Checkout API

You can use the `Business::PayPal::NVP` module to support the most popular and commonly used feature available on the PayPal network: the Express Checkout. You require some logic in your Perl CGI script to manage the different states of the Express Checkout, as multiple API methods need to be used.

The Express Checkout API has four unique stages: introduction, redirection, confirmation, and completion. Your code must manage the user through each stage.

The introduction stage uses the API method `SetExpressCheckout`. This introduces PayPal to the user's transaction, allowing you to describe what is being purchased and for how much. This returns a token value that is your unique identifier to this Express Checkout session.

The second stage, redirection, requires you to send the user and token to a special Express Checkout URL, hosted by PayPal. The user authenticates into their PayPal account, confirms the items being purchased and the payment method, and approves the transfer.

For the third stage, confirmation, the PayPal Express Checkout Web site redirects the user back to your Perl CGI script, delivering the original token and a *PayerID* in the query-string URL. The PayerID is your reference to the user's PayPal account. You may optionally create one more confirmation screen, using the `GetExpressCheckoutDetails` method.

The final stage, completion, requires you to close the Express Checkout session by calling `DoExpressCheckoutPayment` with the token and PayerID. This actually executes the transfer of funds and completes the Express Checkout process.

Use the PayPal Express Checkout API

- 1 Open a Perl CGI script that uses `Business::PayPal::NVP`.
- 2 Assign a hard-coded purchase price.

Note: This is temporary, as described later.

- 3 Construct the framework for the four Express Checkout stages.
- 4 Create the starting Express Checkout form that begins the process.
- 5 Type `my %resp = $pp->SetExpressCheckout();`
- 6 Insert the purchase amount and describe the item or items purchased.
- 7 Assign the `RETURNURL` to use this Perl CGI script's URL.
- 8 Assign the `CANCELURL` to use a static HTML page.
- 9 If the method succeeded, redirect the user to the PayPal Web site.
- 10 Include the token returned by `SetExpressCheckout` in the PayPal URL.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
paypal_express-checkout.pl
File Edit Tools Window Help
6 use Business::PayPal::NVP;
7
8 my $cgi = new CGI;
9 my $pp = Business::PayPal::NVP->new( test => { version => 63.0,
10     user => "store1_1277260120_biz_api.array.org", pwd => "1277260125",
11     sig => "An4HWPi66WtA0xT30T4v4EdhsWVAYuZP-ob06LHv02aygpIPB9-Dh",
12     } );
13
14 # Hard-code the purchase amount (for now).
15 my $amount = '42.95';
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

```
24 elsif ( $cgi->param( 'cmd' ) eq 'Checkout' ) {
25     # The user has clicked our checkout button. Start the ExpressCheckout process.
26
27
28
29
30
31
32
33
34
35
36 if ( $resp{ 'ACK' } ne 'Success' ) {
37     print "Content-type: text/html\n\n";
38     print "Failed to run SetExpressCheckout!\n\n";
39
40     print "Debug output of SetExpressCheckout:<br>\n";
41     print "<pre>" . Dumper( \%resp ) . "</pre>";
42 }
43 else {
44     # Redirect the user's browser to the PayPal ExpressCheckout Web site
45     $resp{ 'TOKEN' }
46 }
47 }
48 else {
49     # The user is ready to begin the ExpressCheckout process!
50     print "Content-type: text/html\n\n";
51     print "Form method: post action: '$ENV{ 'SCRIPT_NAME' }'\n\n";
52     print "Purchase amount: $amount\n";

```


PayPal provides you with the token and the PayerID.

- 11 Store the PayPal token and PayerID CGI values into Perl variables.
- 12 Type **my %resp = \$pp->GetExpressCheckoutDetails (TOKEN => \$token);**
- 13 If the method succeeded, create a new form.
- 14 Include the token and PayerID values as hidden form inputs.
- 15 Create a Pay button.
- 16 Type **my %resp = \$pp->DoExpressCheckoutPayment();**
- 17 Include the token and PayerID values in this method.
- 18 Confirm the payment action, amount, and currency.
- 19 If the method succeeds, print a nice Thank You message.
- 20 Save the Perl CGI script.

```

19 # If the token or PayerID have been provided, (typically by PayPal during the
20 # RETURNURL callback) store these values for access later.
21 my $token = $cgi->param( 'token' );
22 my $payerid = $cgi->param( 'PayerID' );
23
24 if ( $cgi->param( 'cmd' ) eq 'Pay' ) {
25     # Complete the ExpressCheckout transaction.
26 }
27 elsif ( $token && $payerid ) {
28     # User has returned from PayPal's ExpressCheckout servers. Validate their
29     # current status to confirm that the checkout process is still active.
30     my %resp = $pp->GetExpressCheckoutDetails( TOKEN => $token );
31
32     print "Content-type: text/html\n\n";
33     if ( $resp{ 'ACK' } ne "Success" ) {
34         print "Failed to run GetExpressCheckoutDetails!\n";
35     }
36     else {
37         print "Please pay for your order!\n";
38         print "<form method=get action='$_ENV{ 'SCRIPT_NAME' }'>\n";
39         print "<input type=submit name=cmd value='Pay'>\n";
40         print "</form>\n";
41     }
42     print "Debug output of GetExpressCheckoutDetails:<br>\n";
43     print "<pre>" . Dumper( { %resp } ) . "</pre>\n";
44 }

```

```

19 # If the token or PayerID have been provided, (typically by PayPal during the
20 # RETURNURL callback) store these values for accessing later.
21 my $token = $cgi->param( 'token' );
22 my $payerid = $cgi->param( 'PayerID' );
23
24 if ( $cgi->param( 'cmd' ) eq 'Pay' ) {
25     # Complete the ExpressCheckout transaction.
26     my %resp = $pp->DoExpressCheckoutPayment(
27         PAYMENTACTION => 'ExpressCheckoutPayment',
28         TOKEN => $token,
29         PAYERID => $payerid,
30         AMT => $amt,
31         CURRENCY => $currency,
32     );
33
34     print "Content-type: text/html\n\n";
35     if ( $resp{ 'ACK' } ne "Success" ) {
36         print "ERROR: Failed to run DoExpressCheckoutPayment!\n";
37     }
38     else {
39         print "Thank you!\n";
40     }
41
42     print "Debug output of DoExpressCheckoutPayment:<br>\n";
43     print "<pre>" . Dumper( { %resp } ) . "</pre>\n";
44 }
45 elsif ( $token && $payerid ) {
46     # User has returned from PayPal's ExpressCheckout servers. Validate their
47     # current status to confirm that the checkout process is still active.
48     my %resp = $pp->GetExpressCheckoutDetails( TOKEN => $token );

```

Extra

Unfortunately, you have to hard-code the purchase price directly into this script because of the simplicity of the example program. PayPal requires you to submit the purchase amount for both the `SetExpressCheckout` and `DoExpressCheckoutPayment` methods, but your program lacks any way of keeping track of this value between those two calls. There is a chance that the user will add shipping and insurance, or apply a discount when using the Express Checkout Web site, and this may affect the final price.

You could use the token and store it into an internal database immediately after `SetExpressCheckout`. When the user returns, you can retrieve it from that database. Alternatively, you could also get the final purchase value out of `GetExpressCheckoutDetails`.

Ideally, you should generate an invoice order number and assign it to the Express Checkout session through the `INVNUM` field, but you still need to link this number to the purchase price using a persistent database. The best possible solution is to set up a relational database, using a program such as MySQL, as described in Chapter 21.

continued

Use the PayPal Express Checkout API (continued)

The PayPal Express Checkout methods allow for dozens of additional fields that you can use to customize how the end-user experiences the process. You can only apply these fields to the `SetExpressCheckout` method, the first stage of the Express Checkout process.

You must set the `AMT` field to establish the total cost of the transaction. You can also set the `CURRENCYCODE` field to establish the base currency. The currency code must be represented in exactly three characters, for example, USD, CAD, or GBP. If you omit the currency code, it defaults to USD. You can also set additional fields to provide more details on how the final amount is calculated. `ITEMAMT` represents the subtotal, `SHIPPINGAMT` and `HANDLINGAMT` represent shipping and handling costs, and `TAXAMT` represents the taxes charged. The item subtotal, shipping,

handling, and tax amounts, if defined, must all add up to the value used for `AMT`.

Individual items in the order can be summarized by fields that begin with `L_` and end in a number. `L_NAME n` represents the n th item's name in the purchase order. `L_DESC n` is its longer description. `L_AMT n` and `L_QTY n` represent the base item price and the total quantity. The first item in the list should start numbering at zero.

Note that, if defined, the sum of all `L_AMT n` times `L_QTY n` fields must equal `ITEMAMT`, unless there are no item subtotal, shipping, handling, and tax amounts, in which case the total of these fields should equal `AMT`.

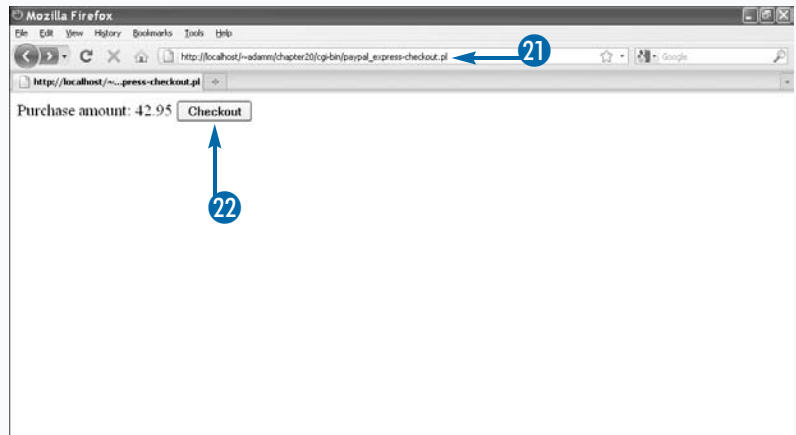
All cost-associated fields must be less than \$10,000.00, and contain up to five digits, one decimal point, and exactly two digits. Do not use commas or any currency characters.

Use the PayPal Express Checkout API (continued)

- 21 Open your Perl CGI script in a Web browser.

The purchase screen appears.

- 22 Click Checkout.

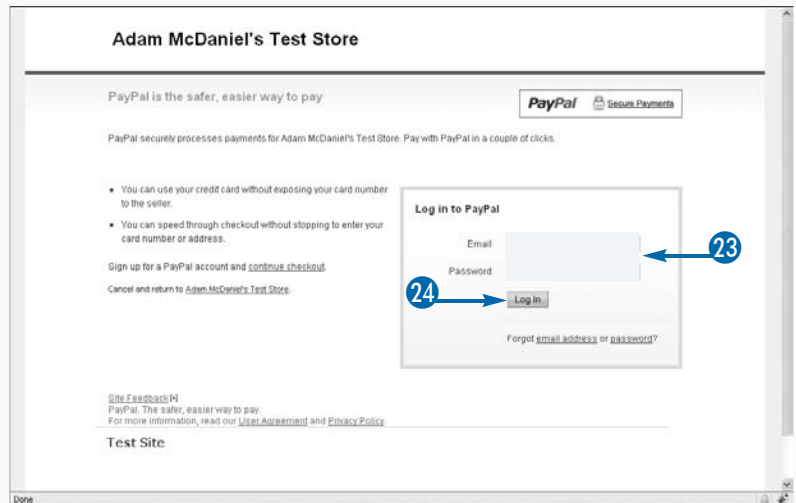


The PayPal Express Checkout Web site loads.

- 23 Type in your e-mail address and password.

Note: Use your sandbox buyer credentials if you are connecting to the PayPal sandbox.

- 24 Click Log In.



The PayPal Express Checkout order summary screen appears.

- The item.
 - The unit price.
 - The quantity.
 - The total price.
- 25 Click Continue.

Description	Unit Price	Quantity	Amount
Add special instructions to merchant			\$42.95
Total:			\$42.95

Payment Method: PayPal Balance

Ship to: Test User, 1 Main Victoria, Toronto Ontario M5A 1E1, Canada

Contact information: user1_127725993_per@array.org

Continue

Your Perl CGI script loads again.

- 26 Click Pay.
- The payment process is completed.

Please pay for your order:

Pay

Debug output of GetExpressCheckoutDetails:

```
$VAR1 = (
  'PAYMENTREQUEST_0_SHIPTONAME' => 'Test User',
  'PAYMENTREQUEST_0_SHIPTOCOUNTRYNAME' => 'Canada',
  'L_ITEM0' => 'Item $5',
  'L_PAYMENTREQUEST_0_ITEMLENGTHVALUE0' => '0.00000',
  'L_TAXAMT0' => '0.00',
)
```

Done

Thank you!

Debug output of DoExpressCheckoutPayment:

```
$VAR1 = (
  'ORDERTIME' => '2010-06-23T09:05:44Z',
  'PAYMENTINFO_0_PAYMENTSTATUS' => 'Completed',
  'SUCCESSPAGEREDIRECTREQUESTED' => 'false',
  'SHIPPINGOPTIONSISDEFAULT' => 'false',
  'REASONCODE' => 'None',
  'TRANSACTIONTYPE' => 'expresscheckout',
)
```

Done

Extra

There are many fields you can specify in `SetExpressCheckout` to further customize the Express Checkout process. Refer to Chapter 6 in the NVP API Developer Guide (ExpressCheckout API Operations) for a complete list of additional request and response fields supported by this method and how to reference it.

FIELD	DESCRIPTION
ALLOWNOTE	Allows the buyer to enter a note to the merchant.
HDRIMG	URL for the header image.
HDRBORDERCOLOR	Sets the border color around the header image.
HDRBACKCOLOR	Sets the background color around the header image.
PAYFLOWCOLOR	Sets the background color for the payment page.
EMAIL	Pre-populates the PayPal login page with this e-mail address.
SOLUTIONTYPE	Type of checkout flow: <code>SOLE</code> for optional PayPal process, <code>MARK</code> for PayPal account required.
BRANDNAME	Overrides the business name in the PayPal merchant account.
SURVEYENABLE	Enables Express Checkout survey functionality.
SURVEYQUESTION	The text for the survey question on the PayPal review page.

Search Your PayPal Transaction History

You can search for all transactions processed by your seller account through the `TransactionSearch` method. This allows you to review and audit your seller account's history using the PayPal network. By default, the only required search field is `STARTDATE`, but you can narrow down the results by searching for matching fields. Additional fields include `ENDDATE`, `EMAIL`, `INVMNUM`, `ACCT`, `TRANSACTIONCLASS`, and `STATUS`.

The response fields that are returned will summarize some information about the transactions, but not everything. Returned fields include `L_TYPEn`, `L_STATUSn`, `L_NAMEn`, `L_AMTn`, `L_FEEAMTn`, and `L_NETAMTn`. You can use `GetTransactionDetails` with the `TRANSACTIONID` field for more information about a specific transaction. Refer to the NVP API Developer Guide document for an explanation of these fields.

Because of the nature of NVP, the returned data will appear in no particular order. There is no sorting done on the PayPal NVP end; fortunately, sorting this yourself is pretty easy in Perl. All fields end in a number to indicate each row returned by the search, and numbering starts at zero. You could use the following code to sort everything into a two-dimensional array reference:

```
my $data = [];  
while ( my ( $key, $val ) = each %resp {  
    $key =~ s/(\d*)$//; $row = $1;  
    $data->[ $row ]->{ $key } = $val;  
}
```

Note that the information returned here will match what is conveyed on the PayPal Merchant Web site when searching for transactions by criteria.

Search Your PayPal Transaction History

- 1 Open a Perl script that uses `Business::PayPal::NVP`.
 - 2 Type `my %resp = $pp->TransactionSearch(`.
 - 3 Type `STARTDATE => "YYYY-mm-dd HH:MM:SS"`.
 - 4 Type `);`.
 - 5 Use `Data::Dumper` to review the contents of `%resp`.
 - 6 Save the Perl script.
 - 7 Open a Terminal window or DOS Prompt.
 - 8 Execute the Perl script.
- The Perl script displays the transactions.
- Two transaction IDs, randomly appearing beside each other.

```
1 #!C:/Perl/bin/perl.exe  
2  
3 use Data::Dumper;  
4 use Business::PayPal::NVP;  
5  
6 my $pp = Business::PayPal::NVP->new( test => {  
7     user => "store1_1277260120_bin_apil.array.org",  
8     pwd => "1277260125",  
9     sig => "An44HFir6zMTA0xYJ0TW4fDH5wIVAYuZf-obX6LHrvG2aygpIpyB9-Dh",  
10    version => 63.0,  
11    } );  
12  
13 my %resp = $pp->TransactionSearch(  
14     STARTDATE => "2010-01-01 12:00:00"  
15 );  
16  
17 print Dumper( { %resp } );  
18
```

```
C:\Documents and Settings\adam\My Documents\My Website\chapter20>  
$VAR1 = (  
    'L_TIMEZONE4' => 'GMT',  
    'L_STATUS4' => 'Completed',  
    'L_AMT6' => '200.00',  
    'L_TYPE1' => 'Payment',  
    'L_TYPE4' => 'Payment',  
    'L_NAME0' => 'Test User',  
    'L_NETAMT4' => '40.97',  
    'L_STATUS3' => 'Completed',  
    'L_NAME2' => 'Test User',  
    'L_CURRENCYCODE3' => 'CAD',  
    'L_TIMEZONE1' => 'GMT',  
    'L_TIMESTAMP2' => '2010-06-23T07:31:05Z',  
    'L_TYPE6' => 'Transfer',  
    'L_FEEAMT6' => '0.00',  
    'L_CURRENCYCODE5' => 'CAD',  
    'L_TIMEZONE3' => 'GMT',  
    'L_STATUS2' => 'Completed',  
    'L_CURRENCYCODE1' => 'CAD',  
    'L_TYPE2' => 'Payment',  
    'L_TYPE0' => 'Payment',  
    'L_EMAIL0' => 'user1_1277259993_per@array.org',  
    'L_TIMESTAMP5' => '2010-06-23T05:55:17Z',  
    'L_CURRENCYCODE6' => 'USD',  
    'L_NAME3' => 'Test User',  
    'L_NETAMT5' => '125.88',  
  
    'L_TIMESTAMP6' => '2010-06-23T02:28:41Z',  
    'L_TIMEZONE0' => 'GMT',  
    'BUILD' => '1362829',  
    'L_NAME6' => 'PayPal',  
    'L_TIMEZONE6' => 'GMT',  
    'L_CURRENCYCODE4' => 'CAD',  
    'CORRELATIONID' => '6ce6ef18707a6',  
    'L_EMAIL1' => 'user1_1277259993_per@array.org',  
    'L_CURRENCYCODE0' => 'CAD',  
    'L_NAME1' => 'Test User',  
    'L_TRANSACTIONID6' => '26F98150KC4428334',  
    ...  
);
```

View a PayPal Transaction's Details

If you are interested in viewing all the information PayPal knows about a transaction, and you already have its transaction ID, then you can use `GetTransactionDetails` to retrieve it using the `Business::PayPal::NVP` module. If you do not already know the transaction ID, you must search for it in your transaction history using the `TransactionSearch` method.

The actual information that appears will vary, depending on the transaction type, its status, and the type of user it affected. For example, a transaction produced using `DoDirectPayment` will not contain as many fields as one produced using `DoExpressCheckoutPayment`. This is because information from the user's PayPal account profile will only appear in the transaction if the account

profile was specifically involved, as it is during the Express Checkout process. A refund transaction will only contain information specific about the refund, and then cite the original transaction ID that it reversed.

Like the other methods that are available on the NVP API, some response fields can only be specified once per transaction; others may be used multiple times, which is indicative of multiple items in the order. In this case, singular fields will be described with their full name, such as `ORDERTIME`, `FIRSTNAME`, `LASTNAME`, `SHIPTONAME`, `SHIPTOCITY`, and so forth. Item-related fields always begin with `L_` and end in a number, such as `L_DESCn`, `L_QTYn`, and `L_AMTn`. If you are parsing the fields with Perl, be aware of this when scanning the results. Note that the information returned here will match what is conveyed on the PayPal Merchant Web site when you are viewing a specific transaction's details.

View a PayPal transaction's Details

- 1 Open a Perl script that uses `Business::PayPal::NVP`.
- 2 Type `my %resp = $pp->GetTransactionDetails(`
- 3 Type `TRANSACTIONID => value,`
- 4 Type `);`
- 5 Use `Data::Dumper` to review the contents of `%resp`.
- 6 Save the Perl script.
- 7 Open a Terminal window or DOS Prompt.
- 8 Execute the Perl script.

The Perl script displays the transaction details.

```

1  paypal_transaction_details.pl
2
3  use Data::Dumper;
4  use Business::PayPal::NVP;
5
6  my $pp = Business::PayPal::NVP->new( test => {
7      user => "store1_1277260120_biz_apil.array.org",
8      pwd  => "1277260125",
9      sig  => "An44HPIr6zWta0xYJ0TW4IDHSwIVAYuIF-obX6LMvv92aygpIpyB9-Dh",
10     version => "1.0"
11 });
12
13 # Get the Transaction ID as the first argument to this program.
14 my $transID = shift;
15
16 my %resp = $pp->GetTransactionDetails(
17     TRANSACTIONID => $transID,
18 );
19
20 print Dumper( \%resp );
21
22

```

```

C:\Documents and Settings\adam\My Documents\My Website\chapter20:
2170939710770053M
$VAR1 = {
    'SHIPTOSTREET' => '1 Maire-Victorin',
    'L_CURRENCYCODE0' => 'CAD',
    'ORDERTIME' => '2010-06-23T08:00:26Z',
    'CUSTOM' => 'Thank you!',
    'SALESTAX' => '0.00',
    'SHIPTOSTATE' => 'Ontario',
    'REASONCODE' => 'None',
    'L_TRANSACTIONID' => '0.00',
    'PAYMENTTYPE' => 'Instant',
    'TRANSACTIONTYPE' => 'expresscheckout',
    'SHIPHANDLERAMOUNT' => '0.00',
    'EMAIL' => 'user1_1277259999_per@array.org',
    'SHIPTONAME' => 'Test User',
    'COUNTRYCODE' => 'CA',
    'SHIPTOCOUNTRYNAME' => 'Canada',
    'FIRSTNAME' => 'Test',
    'ADDRESSSTATUS' => 'Confirmed',
    'SHIPAMOUNT' => '0.00',
    'FEEAMT' => '1.98',
    'SHIPTOZIP' => 'M5A 1E1',
    'PROTECTIONELIGIBILITY' => 'Eligible',
    'TIMESTAMP' => '2010-06-23T08:52:37Z',
    'PAYERID' => 'P9V7BCPGNMVRC',
    'SHIPTOCOUNTRYCODE' => 'CA',
    'L_OTV0' => '1',
    'PAYMENTSTATUS' => 'Completed',
    'ACK' => 'Success',
    'CURRENCYCODE' => 'CAD',
    'PENDINGREASON' => 'None',
    'AMT' => '42.95',
    'RECEIVERID' => 'KDJZ6MXL25HKW',
    'BUILD' => '1362829',
    'ADDRESSOWNER' => 'PayPal',
    'LASTNAME' => 'User',
    'TRANSACTIONID' => '2170939710770053M',
    'SHIPTOCITY' => 'Toronto',

```

Refund a PayPal Transaction

You can issue a refund transaction for any payment transaction processed with your merchant account on PayPal. The refund will reverse funds from your seller account's PayPal balance, and direct it back to the buyer's original funding source. This can happen at any time after a transaction has cleared. To issue a refund, you can use the `RefundTransaction` method with the original transaction ID.

Sometimes, you may want to only issue a refund for a partial amount of the original purchase. This is handled by the request field `REFUNDTYPE`. If you set it to `Partial`, you must also supply `AMT` and `CURRENCYCODE` fields. If you set the type of refund to `Full`, you must omit these fields.

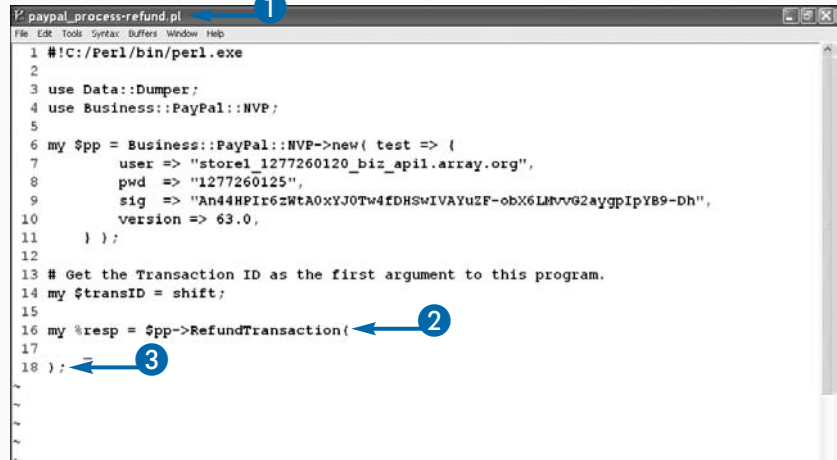
You may also add a `NOTE` field into this refund transaction. This allows you to record a possible reason why the buyer required a refund. PayPal will keep track of this value and display it each time you display this refund transaction's details.

Note that PayPal does charge a fee for refunds. The exact amount depends on your merchant account type and country of origin, but could be as high as five percent of the original transaction amount. PayPal will summarize for you in the response fields the `GROSSREFUNDAMT`, `FEEREFNDAMT`, and `NETREFUNDAMT` values.

In your transaction log, PayPal generates a new transaction ID for this refund, and assigns to it the original purchase transaction ID in the `PAYMENT TRANSACTIONID` field. The original purchase transaction's `PAYMENTSTATUS` field will change from `Completed` to `Refunded`.

Refund a PayPal Transaction

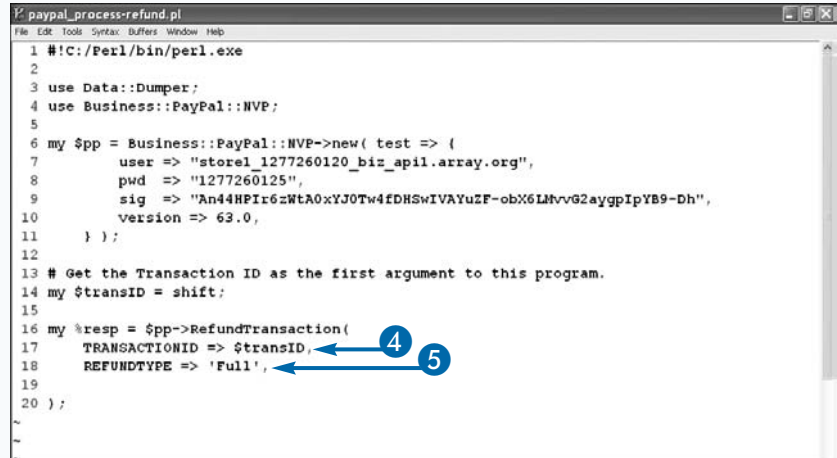
- 1 Open a Perl script that uses `Business::PayPal::NVP`.
- 2 Type `my %resp = $pp->RefundTransaction(`.
- 3 Type `);`.



```
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_apil.array.org",
8     pwd => "1277260125",
9     sig => "An44HPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LhvvG2aygpIpyB9-Dh",
10    version => 63.0,
11 });
12
13 # Get the Transaction ID as the first argument to this program.
14 my $transID = shift;
15
16 my %resp = $pp->RefundTransaction(
17
18 );
```

- 4 Type `TRANSACTIONID => value,`.
- 5 Type `REFUNDTYPE => type,`.

Note: If the refund type is not `Full`, you must also provide the `AMT` and `CURRENCYCODE` fields.



```
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_apil.array.org",
8     pwd => "1277260125",
9     sig => "An44HPir6zWtA0xYJ0Tw4fDHSwIVAYuZF-obX6LhvvG2aygpIpyB9-Dh",
10    version => 63.0,
11 });
12
13 # Get the Transaction ID as the first argument to this program.
14 my $transID = shift;
15
16 my %resp = $pp->RefundTransaction(
17     TRANSACTIONID => $transID,
18     REFUNDTYPE => 'Full',
19
20 );
```


- 6 Type **NOTE** => *text*, to store an optional note with the refund.
- 7 Use **Data::Dumper** to review the contents of `%resp`.
- 8 Save the Perl script.

```

paypal_process-refund.pl
File Edit Tools Syntax Buffers Window Help
1 #!C:/Perl/bin/perl.exe
2
3 use Data::Dumper;
4 use Business::PayPal::NVP;
5
6 my $pp = Business::PayPal::NVP->new( test => {
7     user => "store1_1277260120_biz_api1.array.org",
8     pwd  => "1277260125",
9     sig  => "An44HPir6zWtA0xYJ0T4fDHSwIVAYuZF-obX6LhtvG2aygpIpYB9-Dh",
10    version => 63.0,
11 } );
12
13 # Get the Transaction ID as the first argument to this program.
14 my $transID = shift;
15
16 my %resp = $pp->RefundTransaction(
17     TRANSACTIONID => $transID,
18     REFUNDTYPE => 'Full',
19     NOTE => 'Broken item, returned for warranty repair',
20 );
21
22 print Dumper( { %resp } );
23

```

- 9 Open a Terminal window or DOS Prompt.

- 10 Execute the Perl script.

The refund transaction is processed.

- The PayPal fee for the refund.
- The net refund amount.

```

C:\Documents and Settings\adam\My Documents\My Website\chapter20>
9397T0770053M
$VAR1 = {
    'CURRENCYCODE' => 'CAD',
    'REFUNDTRANSACTIONID' => '0K31283127596164D',
    'FEEREFUNDANT' => '1.98',
    'TIMESTAMP' => '2010-06-23T09:01:51Z',
    'BUILD' => '1366358',
    'GROSSREFUNDANT' => '42.95',
    'NETREFUNDANT' => '40.97',
    'CORRELATIONID' => '1ab78f098c951',
    'VERSION' => '63',
    'ACK' => 'Success';
};
C:\Documents and Settings\adam\My Documents\My Website\chapter20>

```

Extra

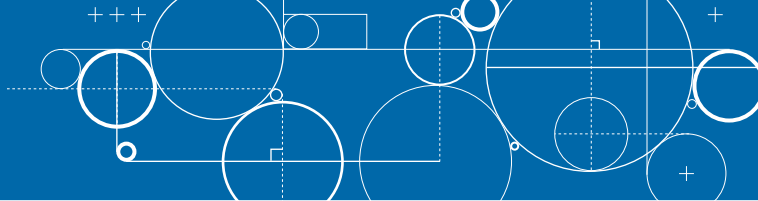
It is possible to issue a refund to a PayPal user if you do not have an original transaction ID. Obviously, from an accounting perspective, you should avoid issuing this type of refund as it will be recorded on your PayPal account as a refund-only transaction. Your accountant may inquire why you are randomly giving money away!

```

%resp = $pp->DoNonReferencedCredit( AMT => amount, NOTE => text,
    CURRENCYCODE => code, CREDITCARDTYPE => cctype, ACCT => ccnumbers,
    EXPDATE => mmyyyy, FIRSTNAME => firstname, LASTNAME => lastname,
    STREET => address, CITY => city, STATE => state,
    COUNTRYCODE => country, ZIP => zipcode );

```

There are other request fields that you can optionally use with this method. Refer to Chapter 5 in the NVP API Developer Guide (DoNonReferencedCredit API Operation) for a complete list of what is available and how to reference it.



Introducing the MySQL Database

MySQL is a relational database system that is available for Linux, Windows, and many other platforms. You can use MySQL to efficiently store and retrieve data for your Perl CGI Web site, when you interface with it using the Perl DBI library.

The core of the MySQL source code has been released under the terms of the GNU General Public License (GPL); however, it still has proprietary components that are licensed separately. For this reason, a free *Community* version of MySQL is available for download, along with a paid *Enterprise* version for higher-end deployments.

For small-scale Web sites, the MySQL client and server are installed directly onto the Web server's computer. This is possible because the server's CPU can comfortably handle the overall load of Apache and MySQL. For larger

Web sites, you should use a dedicated computer to host the MySQL server; however, in this case each Web server requires the separate MySQL client program. In other words, the MySQL client facilitates communications with a local or remote MySQL server.

In this chapter you will learn what MySQL is, how to install it, how basic SQL statements work, how the Perl DBI library interfaces with it, and how you can leverage DBI in your own Perl scripts.

The specific MySQL tasks covered in this book are very limited, compared to the full scope of what MySQL and DBI have to offer you. While MySQL is not the primary focus of this book, it is a very important topic for any Web site that wants to bill itself as professional, dynamic, feature-rich, and user-friendly.

MySQL Server Editions

MySQL comes in two primary editions that vary in the features they offer. Which one you choose depends on your specific needs, the level of support you require, and your budget.

MySQL Community Server The Community edition of MySQL is the free version of the MySQL database software that is licensed under the GPL. As of mid-2010, the latest stable version of MySQL Community Server is 5.1.46, and the development version is 5.5.3. You can download either version directly from the MySQL download page at http://dev.mysql.com/downloads/mysql/ . Generally speaking, unless you require a feature that is available in MySQL 5.5, you should use the stable release of MySQL 5.1. Because you will be using MySQL on your Web site and not redistributing the software with a physical product, you do not need to upgrade to the Enterprise edition.	MySQL Enterprise Server The Enterprise edition of MySQL is the commercial version of the MySQL database software. The commercial entity that manages MySQL Enterprise, MySQL AB, offers additional benefits to entice buyers. These include enhanced server monitoring tools, emergency security patches, monthly maintenance patches, and additional support channels with a guaranteed service level agreement (SLA). If you want to learn more about MySQL Enterprise, visit the Web site at www.mysql.com/products/enterprise/server.html .
--	---

MySQL Server

The MySQL server is the server-side component that hosts the database, manages incoming connections from clients, and interprets ANSI SQL (1999) queries and statements.

Authentication All MySQL clients that connect to your MySQL server need to authenticate themselves to be able to access the data. Often, the server will allow incoming connections from clients on the local server only, or, at most, the local network. If you are running multiple Web sites, or multiple people share your MySQL server, it is a good idea to create user accounts	for each individual site and person, and to restrict access to databases and tables to only those who need it. You need to create a specific username and password for your Perl CGI scripts to use when connecting to the MySQL server using the DBI client library. Generally speaking, it is a good idea to restrict Perl write access as much as possible. This helps reduce the risk of a person visiting your Web site and, purposefully or not, causing an event that detrimentally updates or deletes data on your system.
--	---

MySQL Server (continued)

Performance

The developers of MySQL pride themselves on its enhanced level of performance, compared to its competitors. For instance, MySQL automatically implements query caching, efficient indexing and searching, customizable stored procedures, and high-availability replication.

MySQL efficiently manages its own internal system resources, taking full advantage of multi-processor, high-end servers. Multi-server instances are possible through a master/slave relationship, or by using a newer feature called MySQL Cluster. This feature divides the database into semi-redundant components, and assigns each node in the cluster specific components. In fact, MySQL Cluster does for databases what RAID does for file systems.

MySQL Clients

MySQL allows for client-side programs to interact with its server-side components. Once a client has authenticated itself to the MySQL server, it may execute various SQL commands to interact with the database.

Command-Line Interface

The MySQL command-line interface is a program that runs in either a DOS prompt or a Terminal window. It allows you to connect to a MySQL server and execute SQL statements by typing them in. It is the most direct way to interact with a SQL database.

When developing a Perl script that references SQL statements, you should keep the command-line interface client open in the background. If you experience any syntax issues in Perl, you can test the equivalent statement in the command-line interface; you may notice an additional syntax error messages that were invisible from the perspective of Perl.

the MySQL command-line interface program; however, the DBI library does provide additional conveniences that are not available in other clients.

This chapter is dedicated to describing how to use the Perl DBI library to interact with a MySQL server. For more information about DBI, see the section, "Introducing the Perl DBI Library."

Graphical User Interface

The MySQL developers produce an official graphical user interface, or GUI, called MySQL Workbench. This helps you with database design, SQL statement development, and server administration. All database statements and commands can be represented as graphical operations, which makes using MySQL easier for beginning SQL developers.

Several third-party developers have produced their own GUI programs, such as MySQL Administrator and MySQL Query Browser.

Perl DBI Library

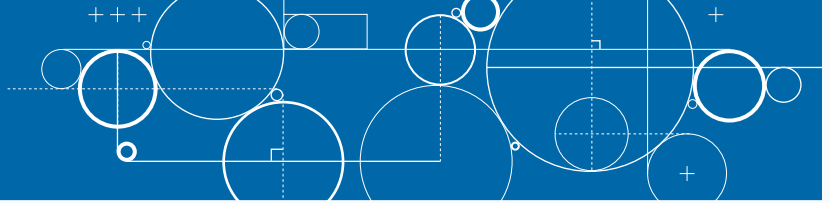
The Perl DBI library provides the means for a Perl script to communicate with a MySQL server as a client. Once you connect to a MySQL database in Perl, the commands you use to interact with the database are exactly the same as

MySQL Documentation and Resources

MySQL has been a staple tool since it was first released in 1995. Since then, it has been well documented by its developers, as well as third-party users and authors. The MySQL 5.1 and 5.5 Reference Manuals are available online at <http://dev.mysql.com/doc/>.

If you are just starting out, useful resources include *Chapter 3: Tutorial*, *Chapter 10: Data Types*, *Chapter 12: SQL Statement Syntax*, and *Appendix A: MySQL 5.1 FAQ*. Another excellent resource for MySQL development is the *MySQL Administrator's Bible* (Wiley 2009).

Understanding the SQL Syntax



SQL, or Structured Query Language, is a *fourth-generation* programming language: you instruct the computer *what* you want a task to produce as output. By comparison, Perl is a *third-generation* language: you instruct the computer *how* a task is to produce the output. This means you are telling the computer what you want to see, as opposed to instructing the computer how to actually do it.

An individual SQL statement is a phrase that begins with a specific verb, followed by a series of options and arguments, and ending with an optional clause; a valid statement almost reads like a normal sentence. The SQL server's parsing abilities are very particular, so each

statement must follow SQL standards exactly. It is not very forgiving if even one word or comma is off.

Note that SQL is an immense language that has generated hundreds of books and dozens of server implementations. The commands listed here are merely a subset of the complete list, and should be enough to get you started with MySQL in Perl.

If you need help constructing a complex or advanced SQL statement, try using a program such as MySQL Workbench or MySQL Query Browser. You can describe your task using its graphical interface, after which the equivalent SQL statement is presented to you along with the results. This is a very useful technique to discover new SQL statements and results.

Basic SQL Statements

The following commands provide the foundation for communicating to a SQL database. At a minimum, you need to be familiar with these commands because they are required to read and write data in the database.

Create a Database

After installing a fresh MySQL server, the first thing to do is to create a new database:

```
create database DATABASE
```

A database houses the tables that store the actual data as rows. In the context of a simple, dynamic Web site, the entire site should reference data from multiple tables, but out of a single database. More complex dynamic Web sites may pull information from multiple databases, as required.

Connect to a Database

Once a database is established, you need to instruct the MySQL client that all future commands are to be applied to that specific database:

```
use DATABASE
```

When using the command-line MySQL client, you must use this command only immediately after establishing a connection to the server. When using the Perl DBI module, you specify the database in the DSN field of your script. See the section, "Introducing the Perl DBI Library," for an example.

Create a Table

Every table within the database has a *schema*, or format, describing each column. After the table is created, you can insert, select, update, and delete data from it by table row:

```
create table TABLE ( COLUMN1 DATATYPE1, COLUMN2  
DATATYPE2, ... )
```

Each column has a data type, which dictates the type of data that is stored in the column. For numerical values, use `int` for a whole number and `float` for a floating-point number. For a specific timestamp, use `date`, `time`, or `datetime`. For strings, `varchar(##)` is most common, where the number represents the maximum number of characters.

Every table should have a column for an auto-incrementing, uniquely identifying number. This value increases with each new row that is inserted, and it allows you to isolate an individual row. The identifier's integer data type also has additional properties: `primary key` indicates that a value cannot be repeated in another row, `auto_increment` states that its value increases by one from the previous row, and `not null` indicates that it cannot be undefined:

```
create table TABLE ( id int primary key auto_  
increment not null, COLUMN2 DATATYPE2, ... )
```

Note that this list of data types is not complete. Refer to the MySQL documentation for the complete list of data types, and for the commands to update and delete an existing table.

Basic SQL Statements *(continued)***Insert Data into a Table**

When inserting a new row of data into a table, you need to specify all columns first, followed by all values. For the columns that you do not specify, the column's default value is auto-assigned into that row:

```
insert into TABLE ( COLUMN1, COLUMN2, ... )
values ( VALUE1, VALUE2, ... )
```

If your table's schema is fairly small, you can bypass stating the columns, which implies them all, and specify all column values:

```
insert into TABLE values ( VALUE1, VALUE2, ... )
```

Select Data from a Table

You can use `select` as the primary means to retrieve data from a table. The `where` clause allows you to only retrieve specific rows that match a condition. If you omit `where`, all rows in the table are returned:

```
select COLUMN from TABLE where CONDITION
```

To retrieve multiple columns from a table, you can specify them as a comma-separated list:

```
select COLUMN1, COLUMN2, ... from TABLE where
CONDITION
```

You can use an asterisk (*) to return all columns in the table that match the condition:

```
select * from TABLE where CONDITION
```

Update Data in a Table

You can use `update` to change an existing row in a table. The `where` clause allows you to limit which rows will be affected. If you omit `where`, all rows in the table will be updated:

```
update TABLE set COLUMN = VALUE where CONDITION
```

You can update multiple columns by separating each pair with a comma:

```
update TABLE set COLUMN1 = VALUE1, COLUMN2 =
VALUE2, ... where CONDITION
```

Delete Data from a Table

The `delete` command removes all rows in a table that match a specific condition:

```
delete from TABLE where CONDITION
```

Be careful — if you omit the `where` clause, every row in the table is deleted without confirmation or warning.

Advanced SQL Statements

Advanced SQL statements are like compounded basic statements. Once constructed, they allow you to efficiently perform a complex task on the SQL database using a single command statement. Again, the following list is just an introduction to what is available. For more information, see the MySQL Reference Manual.

Join Tables

You can select content from multiple tables at once using a single SQL statement. This can be useful if two tables share a column relationship. When you reference more than one table, there must be a `where` clause that describes the relationship:

```
select TABLE1.COLUMNS, TABLE2.COLUMNS from
TABLE1, TABLE2 where TABLE1.COLUMN1 = TABLE2.
COLUMN1
```

In the MySQL Reference Manual, see section 12.2.9.1 for more information on joining tables.

Subquery Statements

A subquery statement allows you to nest a second `select` statement within the first statement's `where` clause. All values that match the second statement are used as matching values for the first statement:

```
select COLUMNS from TABLE1 where COLUMN1 =
( select COLUMN1 from TABLE2 )
```

In the MySQL Reference Manual, see section 12.2.10 for information on constructing subquery statements.

Download MySQL for Windows

You need to download MySQL from the MySQL Web site and install it onto your Web server. This process sets you up with both the MySQL server and client software components simultaneously. If you have multiple Web servers, you may share a central database by installing MySQL server on a dedicated database server, and MySQL client on each Web server.

You want to download the MySQL Community Server edition. This version is freely available for most projects and you can easily upgrade it. If you are unsure whether your Web site can appropriately use the free version, contact a MySQL sales representative at www.mysql.com/about/contact/sales.html.

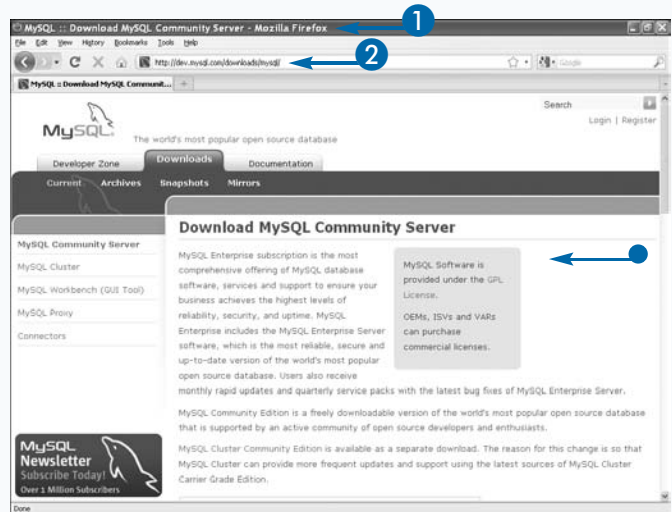
The latest stable release of the MySQL server is version 5.1, and the latest development version is 5.5. You can

download both versions from the MySQL Web site at <http://dev.mysql.com/downloads/mysql/>. The stable version is available under the Generally Available (GA) Releases tab, and the development version is under the Development Releases tab. Unless you are interested in a specific feature in version 5.5, you should download the stable 5.1 version. If you just want the MySQL client and server programs, you can download the Essentials installer. If you require the entire MySQL suite, including the Instance Manager tool, documentation, and various developer components, then you can download the larger Full installer.

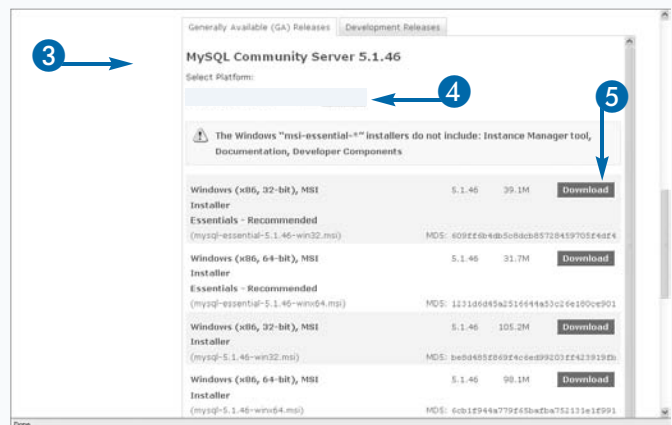
When downloading MySQL for Windows, the software is packaged as an MSI, or Microsoft Installer File. You have the option to download either the 32-bit or 64-bit version of the program. Unless you know that you are running a 64-bit version of Windows, you should download the 32-bit installer.

Download MySQL for Windows

- 1 Open a Web browser.
 - 2 Type <http://dev.mysql.com/downloads/mysql/>.
- The MySQL Community Server download page appears.

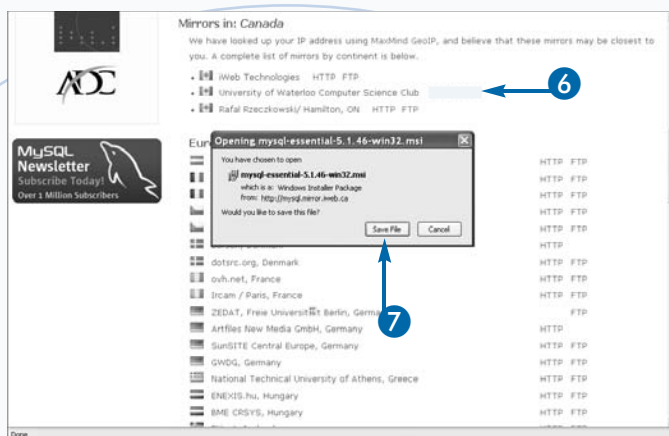


- 3 Scroll to the platform selection box.
- 4 Choose Microsoft Windows from the drop-down list and click Select.
- 5 Click the Download button for the Essentials Installer.

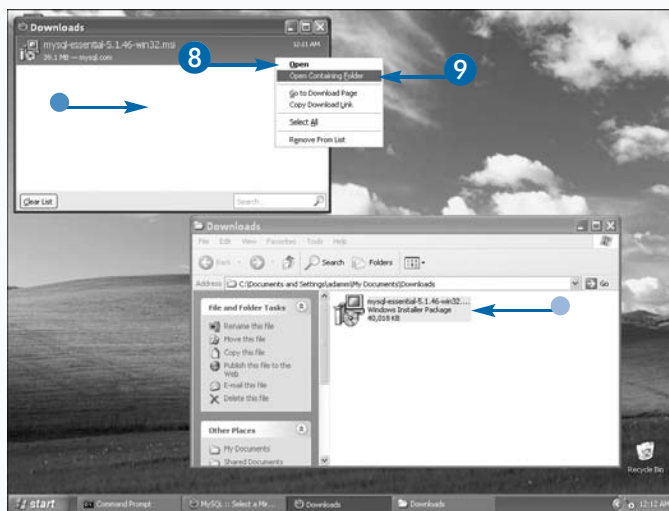


Note: You have the option to create an account on mysql.com, or to scroll down to download anonymously.

- 6 Click either the HTTP or FTP link next to a mirror.
- 7 In the dialog box that appears, click Save File.



- The MySQL Installation MSI is downloaded.
- 8 Right-click the download.
 - 9 Select Open Containing Folder.
- The MySQL MSI installation program.



Extra

When downloading the MySQL server, you can also download additional programs depending on what options you choose. If you choose the full download, you also receive the Instance Manager tool, a local copy of the documentation, and special developer components. In most cases, unless you are planning on running multiple instances of the server, cannot access the online documentation, or are developing a new feature for MySQL, you should download the Essentials version of the server.

Other programs are also available for free on the MySQL download page:

- **MySQL Cluster** is like the MySQL server, except the database is shared across multiple physical machines, also known as nodes, thus providing constant access. Use this only if you have a large database and a lot of user activity, and when you cannot afford any downtime.
- **MySQL Connectors** allows you to link a legacy ODBC- or JDBC-compatible software program into MySQL. This is only necessary if you already have a program written in Java, .NET, C, or C++ connecting to an ODBC or JDBC database and you want to migrate it to MySQL.
- **MySQL Workbench** is a graphical user-interface program that you can use to design tables, execute queries, and administer your MySQL server from the Windows desktop.

Install MySQL for Windows

You need to install MySQL alongside your Perl and Apache Web server so that you can use it as a database back-end for your Perl CGI Web pages. This usually means installing it onto the same server that hosts Perl and Apache, or onto a separate server in the same subnet as your Web server. In a multi-server production deployment, only the MySQL client should be installed onto every Web server alongside Apache and Perl. You should configure each client to connect to a centralized MySQL server. It is inefficient to install the full MySQL server onto each Web server, as each machine's database will be autonomous, unless you enable replication. Instead, you may want to consider using MySQL Cluster, which is better suited for multi-server scenarios.

The Windows installation wizard displays several screens, guiding you through the installation process. Once

Install MySQL for Windows

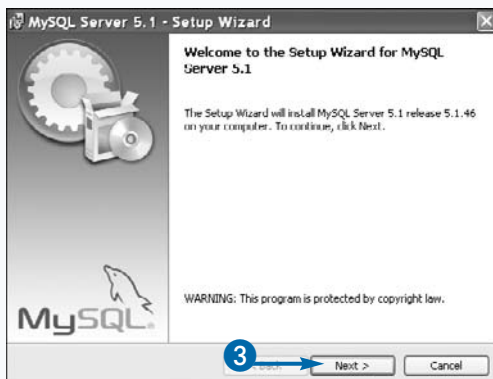
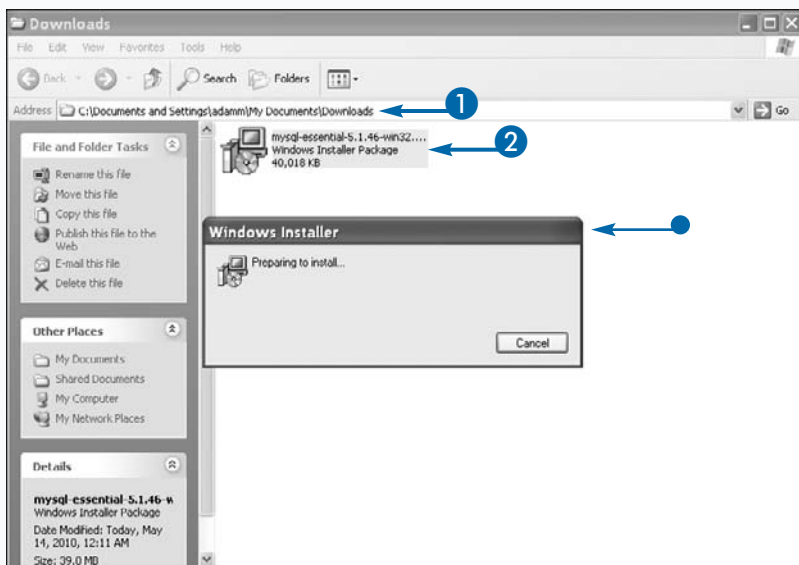
- 1 Open the Downloads folder.
- 2 Double-click the MSI file.
 - The installation program launches.

- 3 In the Setup Wizard, click Next.
- 4 Select the Typical Installation option.
- 5 Click Next.
- 6 Click Install.

installed, a separate configuration wizard appears. This prompts you for various configuration and security options, one of which is the ability to set the password for your MySQL administrator user, called root. After installation and configuration are complete, you still need to set up your table schema and specific accounts for your Perl CGI scripts. You can do this by following the MySQL tutorial at

<http://dev.mysql.com/doc/refman/5.1/en/tutorial.html>.

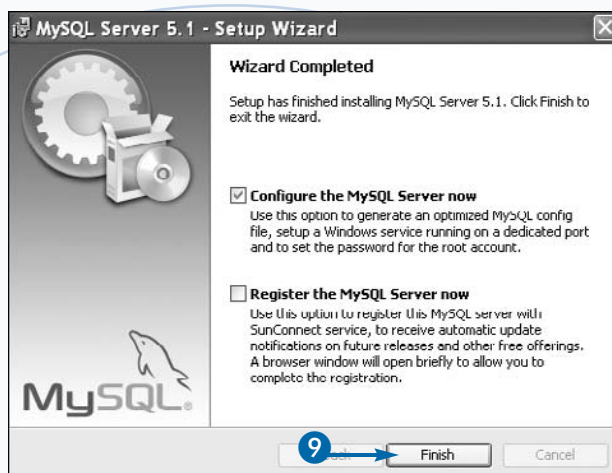
MySQL runs on all recent Windows operating systems, including workstations running Windows 2000, XP, Vista, and 7. However, you should only install MySQL onto a workstation for testing and development purposes. When you are ready to deploy your work onto the Internet, be sure to use MySQL on a Windows Server 2003 or 2008 computer, or even a professionally managed Linux server. MySQL supports both 32-bit and 64-bit native architecture.



- The Setup Wizard installs MySQL.
- 7 Click Next to skip through the MySQL Enterprise advertisement.
- 8 Select the Configure the MySQL Server check box.

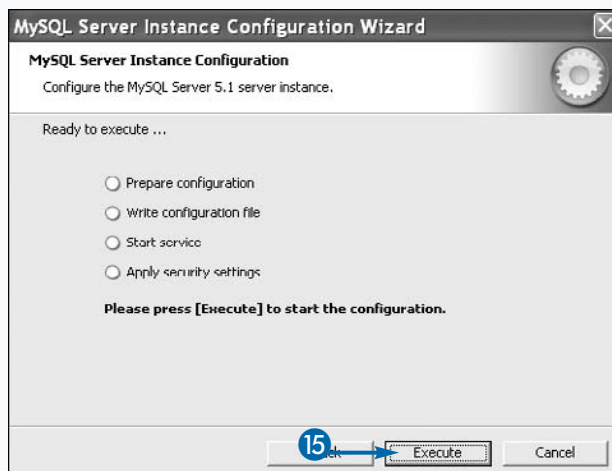
Note: Registration is optional.

- 9 Click Finish.



- 10 Click Next.
- 11 Select the Standard Configuration option.
- 12 Click Next.
- 13 Type in a root password.
- 14 Click Next.
- 15 Click Execute.

The MySQL configuration wizard applies your requested settings to your MySQL server.



Apply It

You must complete the MySQL configuration wizard before you can use your MySQL server. It first prompts you for a configuration type; choose Standard Configuration. You see a check box labeled *Include Bin Directory in Windows PATH*; make sure this is enabled. This allows you to open the MySQL client with the command `mysql` from any directory in a DOS prompt.

The MySQL installation process does not install the Perl Database Interface (DBI) or Database Driver (DBD) modules, which you will require later in this chapter. If you are using ActiveState Perl, you can install these modules from the command-line.

TYPE THIS

```
ppm install DBI
ppm install DBD:mysql
perldoc DBI
perldoc DBD:mysql
```



RESULTS

The Perl DBI and DBD modules are installed using ActiveState Perl Package Manager. Using the PerlDoc program, you can bring up each module's documentation to validate that they are present on your computer.

Install MySQL for Debian/Ubuntu Linux

The installation of MySQL server is very easy on Linux when using a pre-packaged binary distribution system. You can launch the download, installation, and configuration process with a single command. Debian and Ubuntu systems use the DEB packaging format. The suite Advanced Package Tool, or APT, is used for downloading, installing, and removing DEB packages. On Debian- and Ubuntu-based Linux systems, the MySQL server DEB package is identified as `mysql-server`, and the MySQL client is `mysql-client`. You need to install both packages.

The installation process requires that your computer be connected to the Internet. The download, installation, and setup process is handled by a single command, `apt-get`, which you should execute in a Terminal window.

You can also install the packages `mysql-admin`, `mysql-query-browser`, and `mysql-workbench-oss` to connect using GUI clients to your local MySQL server.

Regarding documentation, as of Ubuntu v9.10, only the MySQL 5.0 documentation is available, as the package `mysql-doc-5.0`; for some reason, the 5.1 and later documentation is unavailable in APT. Once installed, you can access the HTML files under the path `/usr/share/doc/mysql-doc-5.0`; however, you may find that it is just as easy to go to the MySQL Documentation URL at <http://dev.mysql.com/doc>.

After installation and configuration are complete, you still need to set up your table schema and specific accounts for your Perl CGI scripts. You can do this by following the MySQL tutorial at <http://dev.mysql.com/doc/refman/5.1/en/tutorial.html>.

Install MySQL for Debian/Ubuntu Linux

- 1 Open a Terminal window.
- 2 Type **sudo apt-get install mysql-server** and press Enter.
- 3 Type your password if prompted and press Enter.
- 4 Type **Y** and press Enter.

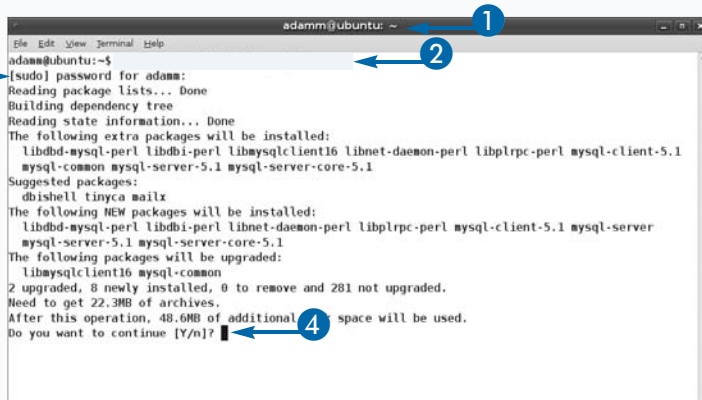
- 5 Type in a root password and press Enter.
- 6 Type in the password again to confirm, and press Enter.

Note: You can change the root password with the command `sudo dpkg-reconfigure mysql-server-5.1`.

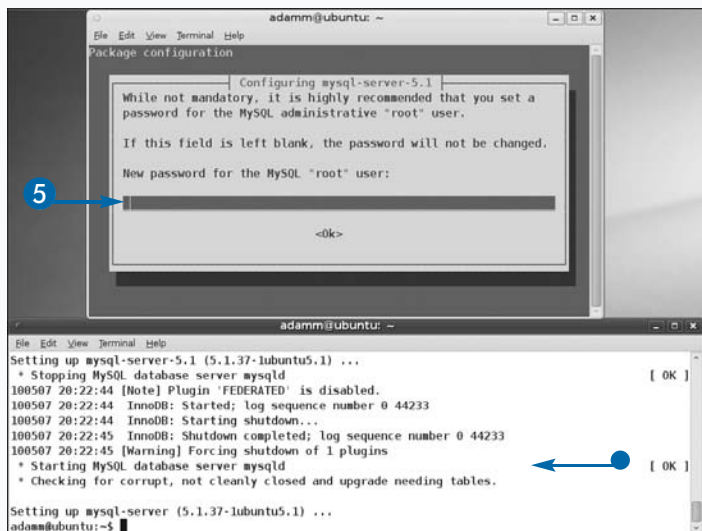
The MySQL installation process continues.

- The MySQL server is now installed.

The DBI and DBD modules are also installed.



```
adam@ubuntu: ~  
File Edit View Terminal Help  
adam@ubuntu:~$ sudo apt-get install mysql-server  
[sudo] password for adam:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following extra packages will be installed:  
  libdbd-mysql-perl libdbi-perl libmysqlclient16 libnet-daemon-perl libplrpc-perl mysql-client-5.1  
  mysql-common mysql-server-5.1 mysql-server-core-5.1  
Suggested packages:  
  dbshell tinyca mailx  
The following NEW packages will be installed:  
  libdbd-mysql-perl libdbi-perl libnet-daemon-perl libplrpc-perl mysql-client-5.1 mysql-server  
  mysql-server-5.1 mysql-server-core-5.1  
The following packages will be upgraded:  
  libmysqlclient16 mysql-common  
2 upgraded, 8 newly installed, 0 to remove and 281 not upgraded.  
Need to get 22.3MB of archives.  
After this operation, 48.6MB of additional space will be used.  
Do you want to continue [Y/n]? 
```



```
adam@ubuntu: ~  
File Edit View Terminal Help  
Package configuration  
Configuring mysql-server-5.1  
While not mandatory, it is highly recommended that you set a  
password for the MySQL administrative "root" user.  
If this field is left blank, the password will not be changed.  
New password for the MySQL "root" user:  
 <Ok>  
adam@ubuntu: ~  
File Edit View Terminal Help  
Setting up mysql-server-5.1 (5.1.37-1ubuntu5.1) ...  
* Stopping MySQL database server mysqld  
100507 20:22:44 [Note] Plugin 'FEDERATED' is disabled.  
100507 20:22:44 InnoDB: Started; log sequence number 0 44233  
100507 20:22:44 InnoDB: Starting shutdown...  
100507 20:22:45 InnoDB: Shutdown completed; log sequence number 0 44233  
100507 20:22:45 [Warning] Forcing shutdown of 1 plugins  
* Starting MySQL database server mysqld  
* Checking for corrupt, not cleanly closed and upgrade needing tables.  
Setting up mysql-server (5.1.37-1ubuntu5.1) ...  
adam@ubuntu:~$ 
```

Install MySQL for Red Hat Linux

The installation of a MySQL server is very easy on Linux when using a pre-packaged binary distribution system. In fact, you can launch the download, installation, and configuration process with a single command. All Red Hat-based Linux distributions contain software packages using the RPM packaging format. You can use the suite Yellowdog Updater, Modified (or YUM) for downloading, installing, and removing RPM packages. On Red Hat-based Linux systems, in the MySQL server RPM package identified as `mysql-server`, the MySQL client is `mysql`. You need to install both packages.

If you have multiple Web servers sharing a central MySQL database, you only need to install the `mysql_server` package on that database server, and the `mysql` package on to your Red Hat-based

Install MySQL for Red Hat Linux

- 1 Open a Terminal window.
- 2 Type `su -` and press Enter.
- 3 Type in root's password and press Enter.
- 4 Type `yum install mysql mysql-server` and press Enter.
- 5 Type `Y` and press Enter.

Web servers. If you only have a single Web server, install both packages.

The installation process requires that your computer be connected to the Internet. You launch the download, installation, and setup process by using a single command, `yum`, which you should execute in a Terminal window.

You can also install the package `mysql-gui-tools`, which sets up various GUI programs for you, including Administrator and Query Browser, which you can use to connect to your local MySQL server. If you want to try MySQL Workbench, you need to download that RPM package separately from <http://dev.mysql.com/download/workbench/>. After installation and configuration are complete, set up your table schema and specific accounts for your Perl CGI scripts by following the MySQL tutorial at <http://dev.mysql.com/doc/refman/5.1/en/tutorial.html>.

```

root@fedora:~# su -
[adam@fedora ~]$
[adam@fedora ~]$ yum install mysql mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
--> Package mysql.1686 0:5.1.45-2.fc12 set to be updated
--> Processing Dependency: mysql-libs = 5.1.45-2.fc12 for package: mysql-5.1.45-2.fc12.1686
--> Package mysql-server.1686 0:5.1.45-2.fc12 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.45-2.fc12.1686
--> Running transaction check
--> Package mysql-libs.1686 0:5.1.45-2.fc12 set to be updated
--> Package perl-DBD-MySQL.1686 0:4.013-2.fc12 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

Package Arch Version Repository Size
--
Installing:
mysql 1686 5.1.45-2.fc12 updates 897 k
mysql-server 1686 5.1.45-2.fc12 updates 8.6 M
Installing for dependencies:
perl-DBD-MySQL 1686 4.013-2.fc12 fedora 153 k
Updating for dependencies:
mysql-libs 1686 5.1.45-2.fc12 updates 1.5 M

Transaction Summary
Install 3 Package(s)
Upgrade 1 Package(s)

Total download size: 11 M
Is this ok [y/N]:
  
```

- The MySQL client and server are now installed.
- The DBI and DBD modules are now installed.

```

root@fedora:~#
updates/prestodelta 13 kB 00:00
fedora/prestodelta 1.3 kB 00:00
Processing delta metadata
Package(s) data still to download: 11 M
(1/4): mysql-5.1.45-2.fc12.1686.rpm 897 kB 00:01
(2/4): mysql-libs-5.1.45-2.fc12.1686.rpm 1.5 MB 00:02
(3/4): mysql-server-5.1.45-2.fc12.1686.rpm 8.6 MB 00:11
(4/4): perl-DBD-MySQL-4.013-2.fc12.1686.rpm 153 kB 00:00
-----
Total 688 kB/s | 11 MB 00:16
warning: rpm: HdrFromFdn: Header V3 RSA/SHA256 signature: NOKEY, key ID 57b6cbb4
updates/gpgkey 3.2 kB 00:00
Importing GPG key 0x57b6cbb4 "Fedora (12) <fedora@redhatproject.org>" from /etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-1386
Is this ok [y/N]: y
Running rpm check debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating      : mysql-libs-5.1.45-2.fc12.1686 1/5
  Installing    : mysql-5.1.45-2.fc12.1686 2/5
  Installing    : perl-DBD-MySQL-4.013-2.fc12.1686 3/5
  Installing    : mysql-server-5.1.45-2.fc12.1686 4/5
  Cleanup      : mysql-libs-5.1.45-2.fc12.1686 5/5

Installed:
mysql.1686 0:5.1.45-2.fc12 mysql-server.1686 0:5.1.45-2.fc12

Dependency Installed:
perl-DBD-MySQL.1686 0:4.013-2.fc12

Dependency Updated:
mysql-libs.1686 0:5.1.45-2.fc12

Complete!
[root@fedora ~]#
  
```


Introducing the Perl DBI Library

You can use the Perl Database Interface, or DBI, library to access information stored on a database server running on your network, alongside Perl and Apache.

DBI actually consists of two parts. First, a Perl module provides the interface for your scripts to use. Second, the database driver, or DBD, is used by DBI to provide access to the database's native API and connection methodology. Together, these DBI components can provide support for many third-party servers, such as SQLite2, SQLite3, Oracle, Sybase, DB2, MS-SQL, and PostgreSQL.

In fact, you can even use DBI to interface to local files and have them act like a database, such as a CSV file,

XBase file, or Excel spreadsheet. DBI can even communicate over database network protocols such as ODBC and LDAP.

In this chapter, you will be installing the DBI and DBD::mysql modules to connect your Perl scripts to a MySQL database. The neat thing about using DBI is that your code does not need to be MySQL-aware at all; DBI and DBD handle all the database-specific nuances, and all your code requires are the relevant SQL statements to access the data.

For more information on Perl DBI, you can access its project Web page at <http://dbi.perl.org>.

Import the DBI Library

You can use the DBI library by importing the DBI module into any Perl script. You may need to first install the DBI library onto your system as it is not included on all Perl distributions.

Module Installation

If you are using a Windows system, you need to install the DBI and DBD::mysql modules manually. If you are using a Linux distribution such as Red Hat, Debian, or Ubuntu, the modules' packages are installed alongside the MySQL server installation package.

As always, CPAN should work on most Perl platforms. If you are using ActiveState Perl in Windows, its Perl Package Manager program, or `ppm`, is preferred.

See Chapter 9 for more information on installing modules using CPAN, ActiveState PPM, or pre-compiled packages, depending on your computing environment.

Documentation

The DBI and DBD::mysql modules provide their documentation locally using the PerlDoc program. You can access it at any time using the command-line program:

```
perldoc DBI
perldoc DBD::mysql
```

DBI's documentation focuses on the Perl API in general. DBD::mysql's documentation describes the MySQL-specific driver and any special conditions pertaining to DBI.

In Your Perl Script...

To load the DBI library, all you need to do is import the module with `use` at the top of your Perl script:

```
use DBI;
```

Unlike other libraries, there are no optional arguments or parameters at this stage. In fact, there is not even a `new` function to call. Instead, you use `connect` to link with a database server, which generates a database-handle. This handle is what you use to access the methods contained in the DBI module.

Interacting with the MySQL Server

Regardless of the server you are connecting to, the commands you use, even SQL statements' syntax, can be relatively generic. Only when opening a connection to the server do you need to specify `mysql` in your script. In fact, if at a later date you choose to change your database server from MySQL to Oracle, for example, you only have to change one line in Perl.

Connecting to MySQL

The DSN, or data source name, is a special string that describes what database on the MySQL server you want to connect to, as well as the server's IP address and port number. If you omit the host and port values, it defaults to localhost and 3306:

```
$dsn = "DBI:mysql:database=database;host=hostname;port=port";
$dbh = DBI->connect( $dsn, username, password );
```

The database-handle, `$dbh`, is then used for all other commands through the remainder of the Perl script, such as retrieving and updating data, and disconnecting from the database.

Reading Data from MySQL

Reading data from MySQL by way of a `select` statement happens in four separate stages: prepare, execute, fetch, and finish. You must represent each stage as a Perl statement.

First, you must use the database-handle's `prepare` method to create a statement-handle variable with your SQL command. The statement-handle is used to access the remaining three stages:

```
$sth = $dbh->prepare( sql );
```

Second, with your new statement-handle, you use `execute` with any optional arguments to run the statement on the MySQL server. The arguments are provided as a comma-separated list, and each item in the list replaces any question-mark characters found in the prepared SQL statement:

```
$sth->execute( values );
```

Third, you retrieve the results from the MySQL server by using repeated calls to a `fetch` method. The `fetchrow_hashref`

method is easiest to use because all rows returned can be accessed as a `hashref` whose keys represent the column names in the database:

```
while ( my $row = $sth->fetchrow_hashref() ) {
    print $row->{ column };
}
```

Finally, you complete the statement by using the `finish` method. This closes any temporary memory handles that could have been allocated when retrieving large amounts of data:

```
$sth->finish();
```

Writing Data to MySQL

You can run SQL `insert` or `update` statements with the `prepare`, `execute`, and `finish` steps described previously, but if you only need to run one basic command that returns no output, you can use the `do` shortcut method, which handles all three at once:

```
$dbh->do( sql, attrs, values );
```

Again, the arguments are optional. This is by far the most efficient way to write data to MySQL.

Disconnecting from MySQL

When your program is finished with its database connection, you use the `disconnect` method to free any active memory:

```
$dbh->disconnect();
```

In the context of a Perl script, the database-handle is automatically destroyed when the script is closed. However, it is considered good practice to clean up after yourself by running this command once you no longer need any MySQL connectivity.

Error Handling

If any DBI method results in a failure, it returns a zero return code and allows you to access the native database's error code and message with `err` and `errstr`, respectively. However, the point where you access these methods depends on how far along you are in the DBI workflow.

For example, if you are just connecting, there is no database-handle or statement-handle yet. You must use `err` and `errstr` as scalars in the DBI module directly:

```
DBI->connect( ... ) || die $DBI::errstr;
```

If the error results from either a database-handle or statement-handle, you can access `err` and `errstr` as methods through the failed handle:

```
$dbh->prepare( ... ) || die $dbh->errstr;
$sth->execute( ... ) || die $sth->errstr;
```

Connect to a MySQL Database with the DBI Library

You can use the DBI library to connect to a MySQL database in Perl. This creates a special database-handle, `$dbh`, which you can use to access the DBI's methods. When finished, you must use the database-handle to terminate the connection to the database server. Before you can connect, you need to ensure that both the DBI library and the MySQL-specific DBD library are installed on your Web server. On Linux, these modules are usually pre-installed when you install MySQL using either a DEB or RPM package. On Windows, you may need to install both modules manually. You can confirm that they are installed by using `Perldoc` to open the DBI or `DBD::mysql` documentation pages. If you cannot find either module, use the `cpan` or `ppm` programs to install them.

To prepare a Perl script for database connectivity, you must import DBI and generate a DSN, or database source

name. This describes which driver to use, the database name, and the server's IP address and port number:

```
use DBI;
$dsn = "DBI:driver:database=database;host=IP;port=number";
```

The host and port fields are optional, defaulting to `localhost` and `3306`. Next, call DBI's `connect` method providing the DSN, as well as the username and password. If successful, a database-handle is returned:

```
$dbh = DBI->connect( $dsn, username, password );
```

When you first installed MySQL, it would have prompted you for a default root account and password; use that account here. Finally, the last thing your program should do is call the `disconnect` method from the database-handle. This frees any memory that is allocated to the database connection: `$dbh->disconnect();`

Connect to a MySQL Database with the DBI Library

- 1 Open a Perl CGI script.
- 2 Type `use DBI;` to import the DBI module.
- 3 Type `my $dsn = "DBI:mysql:database=database";` to create a DSN string.
- 4 Type `my $dbh = DBI->connect($dsn, username, password);` to create a new database-handle.

Note: If you do not yet have an account configured, use the "root" username and password you set up during the server installation.

- 5 Type `;host=host` in the DSN.

Note: If MySQL is installed locally, you do not need a port field. It defaults to `localhost`.

- 6 Type `;port=port` in the DSN.

Note: If you did not change the MySQL service port, you do not need a port field in the DSN. It defaults to `3306`.

- 7 Type `|| die $DBI::errstr;` after connect to catch and display any connect errors, if they occur.

```
1 #!C:\Perl\bin\perl.exe
2
3 use DBI;
4
5 my $dsn = "DBI:mysql:database=test";
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" );
7
8 # Display the Server Name (SQL_DBMS_NAME) property.
9 print "Server: " . $dbh->get_info( 17 ) . "\n";
10
11 # Display the Server Version (SQL_DBMS_VERSION) property.
12 print "Version: " . $dbh->get_info( 18 ) . "\n";
13
```

```
5 my $dsn = "DBI:mysql:database=test;host=localhost;port=3306";
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" );
7
8 # Display the Server Name (SQL_DBMS_NAME) property.
9 print "Server: " . $dbh->get_info( 17 ) . "\n";
10
11 # Display the Server Version (SQL_DBMS_VERSION) property.
12 print "Version: " . $dbh->get_info( 18 ) . "\n";
13
14 || die $DBI::errstr;
```

8 Type `$dbh->disconnect;` to close the database-handle.

9 Save the Perl script.

```
connect_DBI.pl
File Edit Tools Syntax Buffers Window Help
1 #!C:\Perl\bin\perl.exe
2
3 use DBI;
4
5 my $dsn = "DBI:mysql:database=test:host=localhost;port=3306";
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;
7
8 # Display the Server Name (SQL_DBMS_NAME) property.
9 print "Server: " . $dbh->get_info( 17 ) . "\n";
10
11 # Display the Server Version (SQL_DBMS_VERSION) property.
12 print "Version: " . $dbh->get_info( 18 ) . "\n";
13
14 $dbh->disconnect;
15
```

10 Open a Terminal window.

11 Run the Perl script.

- Either the program runs normally, or the Terminal window displays any connect errors.

```
Command Prompt
C:\Documents and Settings\adam\My Documents\My Website\chapter21>
C:\Documents and Settings\adam\My Documents\My Website\chapter21>
```

Apply It

You can build a custom SQL module that automates DBI's `connect` and `disconnect` methods. This can be useful if you have multiple CGI scripts that leverage database connectivity, but you want to amalgamate the database connect procedure into a single location.

Regarding the `get_info` method in the example steps, the arguments 17 and 18 are actually constants for `SQL_DBMS_NAME` and `SQL_DBMS_VERSION`. You can get a complete list of all available driver implementation fields using the following code.

TYPE THIS

```
use DBI::Const::GetInfoType;
while ( my ( $key, $val ) = each
    %GetInfoType ) {
    printf "%40s: %s\n", $key, $dbh-
    >get_info( $val );
}
```



RESULTS

All available server fields are displayed, with their `get_info` values. The `%40s` argument in `printf` forces 40-character spacing with text aligned to the right. This makes the output easier to read. With access to `%GetInfoType`, you can use this as a shortcut to get a specific value using `get_info`:

```
print $dbh->get_info( $GetInfoType{
    'SQL_DATA_SOURCE_NAME' } );
```

Retrieve SQL Data Using the DBI Library

You can use the DBI library to execute *SQL data-retrieval* queries to load data stored in a database table. This process can use to retrieve any database content into your Perl script and on your Web page.

First, use the database-handle and call the `prepare` method to specify your SQL statement, which returns a statement-handle:

```
$sth = $dbh->prepare( sql );
```

Second, with the statement-handle, call the `execute` method. This actually executes your statement on the MySQL server, and prepares the returned results in internal memory:

```
$sth->execute();
```

You may retrieve the number of rows affected by your query with `$sth->rows`.

Third, use a `while` loop to iterate through the rows of data returned by your SQL query. On each pass, use the statement-handle to call the `fetchrow_hashref` method. This returns a hash reference of each row returned by your SQL statement, using the literal column names as keys:

```
while ( my $row = $sth->fetchrow_hashref() ) {  
    print $row->{ column };  
}
```

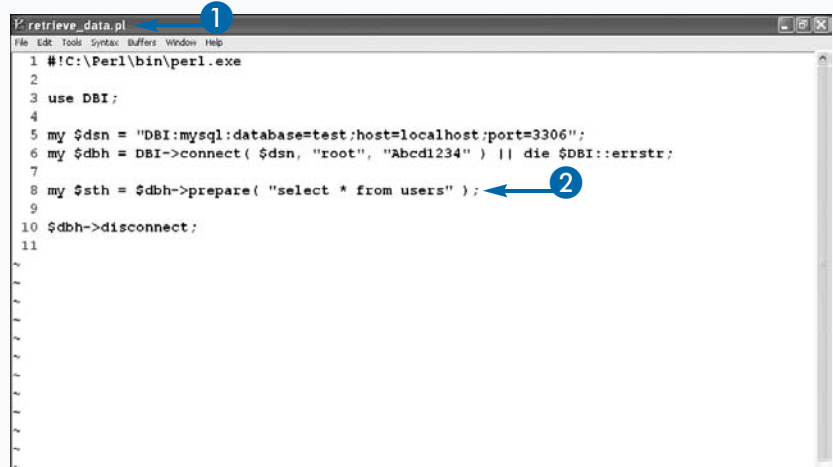
Last, use the statement-handle to call `finish` to instruct the DBI to free any temporary storage data from active memory. `finish` executes automatically when the last row of data has been retrieved with `fetchrow_hashref`. However, you should call it manually to re-execute a previously prepared statement-handle:

```
$sth->finish();
```

Retrieve SQL Data Using the DBI Library

- 1 Open a Perl script that uses the DBI module.
- 2 Type `my $sth = $dbh->prepare (sql);` to create a statement-handle from your SQL query.

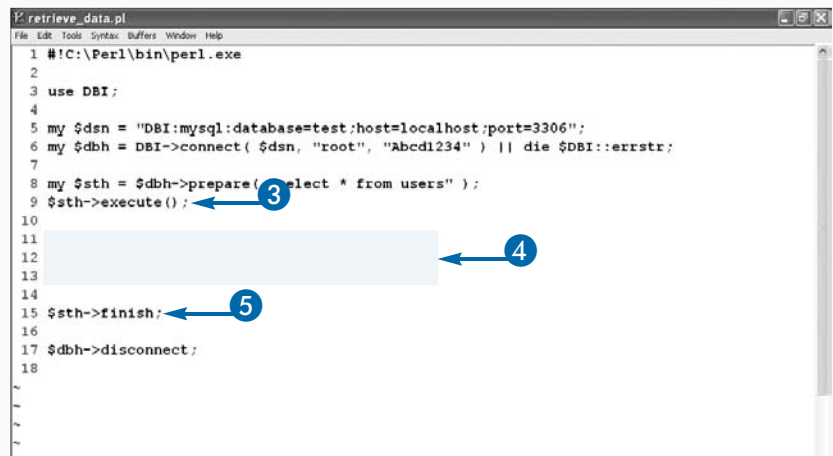
Note: If you have not yet configured a table in MySQL, you have nothing to query. Go through the online MySQL Tutorial at <http://dev.mysql.com/doc/>.



```
1 #!C:\Perl\bin\perl.exe  
2  
3 use DBI;  
4  
5 my $dsn = "DBI:mysql:database=test:host=localhost:port=3306";  
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;  
7  
8 my $sth = $dbh->prepare( "select * from users" );  
9  
10 $dbh->disconnect;  
11
```

- 3 Type `$sth->execute();` to execute the prepared statement.
- 4 Type `my $row = $sth->fetchrow_hashref` within a `while` loop.
- 5 Type `$sth->finish;` to close the statement-handle session.

Note: Calling `finish` manually is optional. It's only necessary if you intend to use `break` within the `while` loop, or to recycle the statement-handle, `$sth`, by calling `execute` again.



```
1 #!C:\Perl\bin\perl.exe  
2  
3 use DBI;  
4  
5 my $dsn = "DBI:mysql:database=test:host=localhost:port=3306";  
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;  
7  
8 my $sth = $dbh->prepare( "select * from users" );  
9 $sth->execute();  
10  
11 while ( my $row = $sth->fetchrow_hashref() ) {  
12     print $row->{ column };  
13 }  
14 $sth->finish;  
15 $dbh->disconnect;  
16
```

- 6 Type `|| die $dbh->errstr;` to catch any prepare errors.
- 7 Type `|| die $sth->errstr;` to catch any execute errors.
- 8 Type `$row->{ column }` to access the actual column data for each row in your query.

Note: This example prints the results as raw Perl output. If this were a CGI script, you would use the `HTML::Template` module to display content to the browser.

- 9 Save the Perl script.
- 10 Execute the Perl script in a Terminal window.
 - The Perl script displays the SQL results.

```

1 #!C:\Perl\bin\perl.exe
2
3 use DBI;
4
5 my $dsn = "DBI:mysql:database=test:host=localhost;port=3306";
6 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;
7
8 my $sth = $dbh->prepare( "select * from users" )
9 $sth->execute()
10
11 while ( my $row = $sth->fetchrow_hashref ) {
12     print "Found username: " . $row->{username} . "\n";
13 }
14
15 $sth->finish;
16
17 $dbh->disconnect;
18
19
20

```

```

C:\Documents and Settings\adam\My Documents\My Website\chapter21>
Found username: testuser
Found username: admin
Found username: guest
C:\Documents and Settings\adam\My Documents\My Website\chapter21>

```

Apply It

You can use one or more question marks (?) in your `prepare` statement as placeholders for bind values supplied as `execute` arguments. This is an extremely important feature as it ensures that your SQL code, when the column values are properly quoted and escaped, is not susceptible to any SQL code-injection attacks.

TYPE THIS

```

use CGI::Carp qw( fatalToBrowser );
$sth = $dbh->prepare( "select * from
  users where createdDate > ?" ) ||
  die $dbh->errstr;
$sth->execute( "2010-05-30" ) || die
  $sth->errstr;

```



RESULTS

The question mark in the SQL query is replaced by the value in `execute`. In other words, the query runs, returning all users created after May 30, 2010. All date-stamps are in YYYY-MM-DD format.

Display SQL Data Through HTML::Template

Using the HTML::Template module, you can efficiently display data sourced from a SQL database, through various template tags. A template file must be constructed referencing the original SQL column names through `TMPL_VAR` tags, which supplies the SQL query results as output. If multiple rows of data are expected, you can use `TMPL_LOOP` to generate a table, within which you can use `TMPL_VAR` to generate each column per output row. To start, you first need to define a blank array reference. This is used to collect the data retrieved from the SQL query, and is then forwarded to the template:

```
my $arrayref = [];
```

The query process is normal, except for the third step. Instead of using the returned hash reference data directly,

you store each row into `$arrayref` with `push`:

```
while ( my $row = $sth->fetchrow_hashref ) {
    push( @{$arrayref}, $row );
}
```

You can use the populated `$arrayref` as a template parameter directly. Because `TMPL_LOOPS` naturally expect an array reference, you can simplify this into a single Perl statement: `$tmpl->param(loopName => $arrayref);`

In your template file, use `TMPL_LOOP` with `loopName`, then `TMPL_VAR` for each returned SQL column:

```
<tmpl_loop name=loopName>
    <tmpl_var name=column1> <tmpl_var
    name=column2> ...
</tmpl_loop>
```

If at any point you are lost on what `$arrayref` holds, you can always use `Data::Dumper` to examine its contents.

Display SQL Data Through HTML::Template

- 1 Open a Perl CGI script that uses the DBI and HTML::Template modules.
- 2 Type `my $arrayref = []`; to create a blank array reference variable.
- 3 Prepare, execute, and retrieve the SQL query normally.
- 4 Print the HTML::Template output normally.

Note: See Chapter 13 for information about using HTML::Template.

- 5 Type `push(@{$arrayref}, $row);` to populate each loop's `TMPL_VAR` values.
- 6 Type `$tmpl->param(listName => $arrayRef);` to populate the template's `TMPL_LOOP` list.

Note: Use `$sth->rows` to get the total number of rows returned.

Note: If you want to bypass `TMPL_LOOP`, you can insert all columns from the first row into the template with `$tmpl->param($sth->fetchrow_hashref);`.

```
1 #!C:\Perl\bin\perl.exe
2
3 use DBI;
4 use HTML::Template;
5
6 my $dsn = "DBI:mysql:database=test:host=localhost:port=3306";
7 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;
8 my $tmpl = HTML::Template->new( filename => "retrieve_data_through_template.tmpl",
9     die_on_bad_params => 0 );
10
11 my $users = [];
12
13
14
15
16
17
18
19
20
21
22 $dbh->disconnect;
23
24
25
26
27 print $tmpl->output;
```

```
1 #!C:\Perl\bin\perl.exe
2
3 use DBI;
4 use HTML::Template;
5
6 my $dsn = "DBI:mysql:database=test:host=localhost:port=3306";
7 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;
8 my $tmpl = HTML::Template->new( filename => "retrieve_data_through_template.tmpl",
9     die_on_bad_params => 0 );
10
11 my $users = [];
12
13 my $sth = $dbh->prepare( "select * from users" ) || die $dbh->errstr;
14 $sth->execute() || die $sth->errstr;
15
16 while ( my $row = $sth->fetchrow_hashref ) {
17     push( @{$users}, $row );
18 }
19 $tmpl->param( userList => $users );
20 $tmpl->param( usersCount => $sth->rows );
21
22 $sth->finish;
23
24 $dbh->disconnect;
25
26 print "Content-type: text/html\n\n";
27 print $tmpl->output;
```


- 7 Open the template file.
- 8 Type `<tmpl_loop name="listName">`.
- 9 Type `<tmpl_var name="column">`.
- 10 Type `</tmpl_loop>`.
- 11 Save the template file.

Note: The content outside of `TMPL_LOOP` is your HTML table formatting; it is only used once. Everything within `TMPL_LOOP` is repeated for each row of the table.

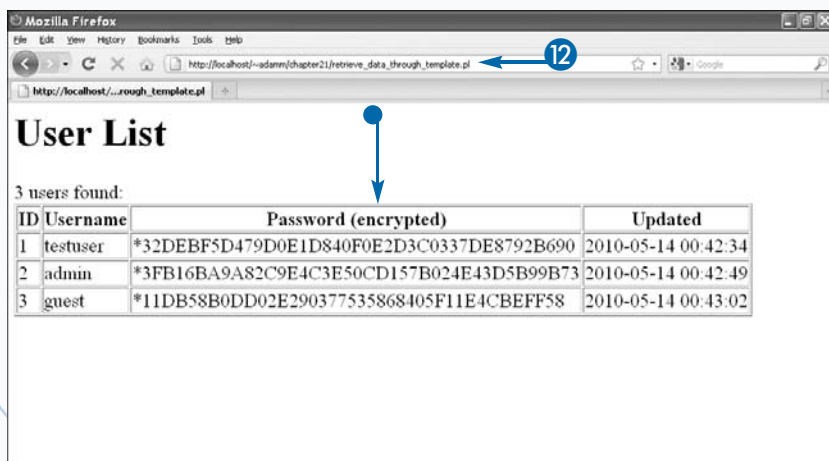
```

1 <h1>User List</h1>
2
3 <tmpl_var name=userCount> users found:<br>
4 <table border=1>
5   <tr>
6     <th>ID</th>
7     <th>Username</th>
8     <th>Password (encrypted)</th>
9     <th>Updated</th>
10  </tr>
11  <tmpl_loop name=userList>
12    <tr>
13      <td>
14      <td>
15      <td>
16      <td>
17    </tmpl_loop>
18 </table>
19

```

- 12 Open the Perl CGI script in a Web browser.

- The HTML table is generated from the template. For every query result row, a table row displays.



Extra

If you are using any built-in MySQL functions in your SQL query, the column name according to the template is literally the function name:

```
select year(date), month(date), min(total) from sales where year(date) = "2010" group by
year(date), month(date);
```

In the template, the `TMPL_VAR` columns are the literal function names specified in the query:

```
Lowest sales in <tmpl_var name="month(date)"> <tmpl_var name="year(date)">: <tmpl_var
name="min(total)">
```

This can become complicated because the names are sensitive to spacing and case. If you were to change the function in the SQL query, you would also have to change the `TMPL_VAR` name to match.

In the SQL query, you can use the `as` modifier to force the column to a specific name. This allows you to change the column name for the duration of the query, and assign a new name independent of the function or formula:

```
select year(date) as salesYear, month(date) as salesMonth, min(total) as lowestSalesTotal from
sales where year(date) = "2010" group by salesYear, salesMonth;
```

The `TMPL_VAR` names can then use the new `as` column name:

```
Lowest sales in <tmpl_var name="salesMonth"> <tmpl_var name="salesYear">: <tmpl_var
name="lowestSalesTotal">
```

Change SQL Data Using the DBI Library

You can use the Perl DBI library to execute SQL data-manipulation statements with the intention of changing the data stored in a database table. Such statements use the SQL commands `update`, `insert`, and `delete`.

You learned that a `select` statement uses the DBI methods `prepare`, `execute`, `fetchrow_hashref`, and `finish` through a statement-handle. Because a data-manipulation statement does not return any query results, you can forgo these four methods in lieu of one: `do`. The `do` method is interchangeable with `prepare`, `execute`, and `finish`. It is best used if you know that there is no requirement for a statement-handle, and you are not retrieving any data from a table:

```
$rows = $dbh->do( sql, attrs, value1,
                 value2, ... );
```

Change SQL Data Using the DBI Library

- 1 Open a Perl script that uses the DBI module.
- 2 Type **`my $rows = $dbh->do(sql);`**.

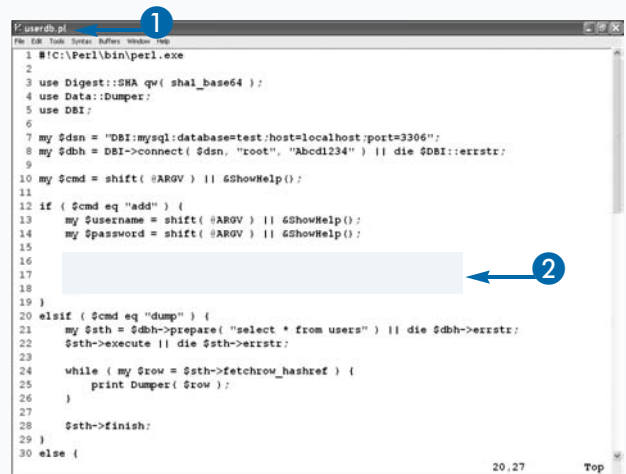
Note: A SQL statement may be on one line, or span multiple lines. In this example, it spans multiple lines to ease readability.

- 3 Insert question-mark characters into the SQL statement.
- 4 Type **`undef`** to specify no additional attributes for `do`.
- 5 Insert the bind values to replace the statement's question marks.

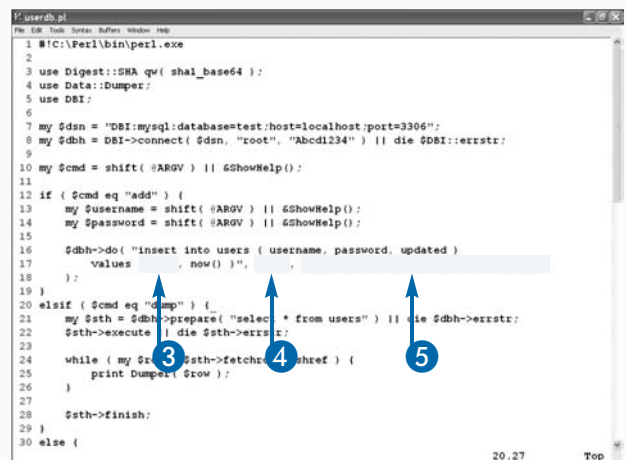
Note: Using bind values automatically escapes any sensitive characters when they are replacing question marks in the statement.

The first argument to `do` is always the SQL statement. The second argument is an attributes hash reference. The remaining arguments represent one or more bind values, which replace any question-mark characters found in the SQL statement. When executed, your SQL statement can apply to multiple rows in your table. You can use `do`'s return value to see how many rows were modified by your statement. If the query is successful but no rows are affected, you may expect `do` to return 0, and if there is a legitimate SQL error, it should return `undef`. When validating its return value, there can be an element of confusion when your code compares zero to undefined.

To resolve this problem, `do` actually returns `"0E0"` if no rows are affected by a good SQL statement, and it returns `undef` if the statement fails. Use `$dbh->err` to find MySQL's native error code, and `$dbh->errstr` to find MySQL's explanation of the error in plain English.



```
1 #!/usr/bin/perl.exe
2
3 use Digest::SHA qw( sha1_base64 );
4 use Data::Dumper;
5 use DBI;
6
7 my $dsn = "DBI:mysql:database=test:host=localhost:port=3306";
8 my $dbh = DBI->connect( $dsn, "root", "Abcd1234" ) || die $DBI::errstr;
9
10 my $cmd = shift( @ARGV ) || $ShowHelp();
11
12 if ( $cmd eq "add" ) {
13     my $username = shift( @ARGV ) || $ShowHelp();
14     my $password = shift( @ARGV ) || $ShowHelp();
15
16     #
17
18 }
19
20 elsif ( $cmd eq "dump" ) {
21     my $sth = $dbh->prepare( "select * from users" ) || die $dbh->errstr;
22     $sth->execute || die $sth->errstr;
23
24     while ( my $row = $sth->fetchrow_hashref ) {
25         print Dumper( $row );
26     }
27
28     $sth->finish;
29 }
30 else {
```



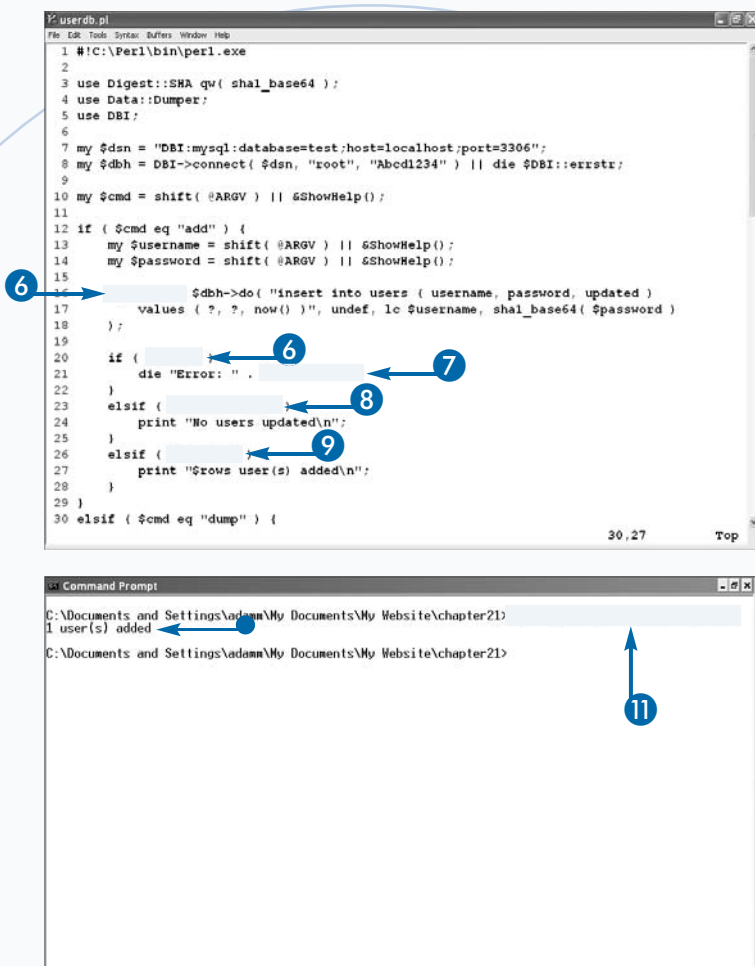
```

31
32     $dbh->do( "insert into users ( username, password, updated )
33             values ( ?, ?, ? )" );
34 }
35
36 elsif ( $cmd eq "dump" ) {
37     my $sth = $dbh->prepare( "select * from users" ) || die $dbh->errstr;
38     $sth->execute || die $sth->errstr;
39
40     while ( my $row = $sth->fetchrow_hashref ) {
41         print Dumper( $row );
42     }
43
44     $sth->finish;
45 }
46 else {
```

- 6 Assign the `$rows` value and test if it is defined.
- 7 If it is not defined, kill the program with `$dbh->errstr`.
- 8 Test if `$rows` is equal to "0E0".
- 9 Test if `$rows` is greater than zero.

Note: You can rewrite the last `elsif` condition as `else`. The test `$rows > 0` is made redundant by the two previous tests: there is no other possible value for `$rows`.

- 10 Save the Perl script.
- 11 Execute the Perl script.
 - The script prints the number of rows that are affected.



Apply It

It is strongly recommended that you use bind values to automatically replace any question mark in your SQL query with properly escaped values. This helps reduce SQL injection attacks as it becomes impossible for a value to supersede the actual statement using strategic quotes and statements within the actual value:

```
$rows = $dbh->do( sql, attrs, value1, value2, ... );
```

It is relatively uncommon to specify any additional attributes in that second argument. They are mostly used to apply special functionality pertaining to the DBI module. As a result, the second argument is often simply represented as `undef`, shifting the starting position for the bind values, which must start as the third argument.

TYPE THIS

```
$dbh->do( "update table set column1 = ?
        where column2 = ?",
        undef, "value1", "value2" );
```



RESULTS

The SQL update statement is executed as `"update table set column1 = 'value1' where column2 = 'value2'".` Had the bind values contained any single- or double-quotes, they would be escaped and treated as literal characters from MySQL's perspective.

Understanding TLS/SSL Encryption

Every modern Web browser supports the two major forms of encryption technology: *transport layer security*, or TLS, and its predecessor, *secure sockets layer*, or SSL. Enabling TLS/SSL encryption on your Web site makes it virtually impossible for a third party to listen in on any session activity between the end-user and the Web server.

In other words, not enabling TLS/SSL encryption means that a clandestine individual could spy on an end-user's HTTP session and record its communication activity. If any sensitive data is transmitted, such as a login

username and password, that data is visible to anyone with access to one of the Internet's core routing hubs, and at both the end-user's and Web site's service provider.

It is not difficult to configure Apache for TLS/SSL encryption, after which users can access your site through an HTTPS URL, such as <https://www.mydomain.com>. The setup procedure requires four steps: creating a private key file, creating a certificate signing request from the key, submitting the request to a certificate authority (CA) to sign, and finally installing the signed certificate and private key using the Apache SSL module.

TLS/SSL Protocol Versions

Over the years, several versions of encryption protocols have been developed, with each new version attempting to address vulnerabilities discovered in its predecessor. The first attempt to encrypt Internet traffic was SSLv1, which was developed in the early 1990s, but never released. SSLv2 became the first public version in 1995, and was able to provide marginal obfuscation through 40-bit and 56-bit encryption. After its release, researchers discovered several vulnerabilities, which prompted the development of SSLv3, featuring 128-bit encryption, in 1996.

Over time, attackers were able to identify flaws in the SSL handshake process, so a new protocol, TLS, was developed. TLSv1 was released in 1999 and has since been implemented as the standard for Internet encryption, providing up to 2048-bit encryption strength.

The Apache SSL module supports SSLv3 and TLSv1. SSLv2 is also available, but is disabled by default. The actual protocol used depends on what the user's Web browser negotiates.

Encryption Prerequisites

Before you can enable TLS/SSL encryption, you must generate a private key and a certificate, and have the certificate signed by a trusted third-party.

Your Private Key

A private key is a special file that consists of a randomly generated series of bytes that are unique to you and your Web site. Generally speaking, your key is like a unique fingerprint and an authorization credential rolled into a single file. You use the private key to create an initial certificate request file, and later, you install the key into Apache to decrypt incoming TLS/SSL traffic.

If your private key were to be obtained by a third party, any certificate you created with it would be compromised. This makes it possible for someone else to decrypt TLS/SSL traffic that is intended only for you.

For this reason, private keys are encrypted using triple DES, and so you must decrypt them each time you use them.

Your Certificate

You use the private key to generate a basic certificate file that describes who you are, where you are located, and the domain name that will be encrypted. The certificate is saved as a certificate signing request, or CSR, which must be submitted to a third party to be signed.

The signing process indicates to your users that your certificate has been validated as real, and that you are who you say you are. Web browsers use your certificate to encrypt data intended only for you. The only way to decrypt the data is with the private key that initially built the certificate.

Encryption Prerequisites *(continued)***The Certificate Authority**

The purpose of the certificate authority, or CA, is to act as a trusted third-party and endorse a Web site implementing TLS/SSL. Because a site's certificate contains identifying

information, users are expected to compare this certificate with the Web site's content. This way, the user can be confident in knowing that their sensitive data, such as their credit card information, can only be read by the Web site, and that the site is authentic.

TLS/SSL on Apache

You can implement TLS/SSL easily in Apache, thus supporting `https://` URLs that end-users can trust as a secure communication channel.

Apache SSL Module

Apache implements TLS/SSL encryption using its SSL Module, commonly referred to as `mod_ssl`. For more information on how `mod_ssl` works, how TLS/SSL encryption works, and for more examples on how to implement it in Apache, read the `mod_ssl` documentation page at http://httpd.apache.org/docs/2.2/mod/mod_ssl.html.

```
SSLCertificateFile PATH
```

Third, you specify the full path to the certificate's private key file:

```
SSLCertificateKeyFile PATH
```

Apache Configuration Directives

The `mod_ssl` library adds dozens of new SSL configuration directives. You do not require all of the directives, but they do allow you to fine-tune how Apache handles TLS/SSL encryption. Apache ships with an example configuration that demonstrates and documents most directives. The location of this file depends on your platform.

PLATFORM	LOCATION
Windows	APACHEDIR/conf/extra/httpd-ssl.conf
Debian/Ubuntu Linux	/etc/apache/mods-enabled/ssl.conf *
Debian/Ubuntu Linux	/etc/apache/sites-enabled/default-ssl *
Red Hat Linux	/etc/apache2/conf.d/ssl.conf

Note that these files are only available after running `a2enmod` and `a2ensite`, respectively. At a minimum, you only need to specify three SSL directives to activate TLS/SSL encryption in `mod_ssl`. Refer to the sample configuration files for additional documentation and examples.

First, you must activate the TLS/SSL engine:

```
SSLEngine on
```

Second, you need to specify the full path to your signed certificate file:

Activate SSL for a Domain

The example Apache SSL configuration is 99 percent complete. All you need to do is specify your unique SSL key and certificate. Once implemented, all traffic coming in on port 443 will implement TLS/SSL. If you only host a single domain, HTTPS encryption is implemented globally on your Apache server.

If you host multiple domains on a single Apache installation, you require a unique SSL certificate for each domain where you choose to implement TLS/SSL encryption. The example SSL configuration is not sufficient, as a signed certificate is domain-specific. Because each domain already requires a valid `<VirtualHost>` configuration group, you can define the domain's SSL directives within, thus limiting the scope of each certificate file to its pre-assigned domain name.

Automatic HTTPS Redirectors

It is possible to force your users into a secure connection, regardless of the URL they use to access your Web page. Using the Apache Rewrite module, commonly known as `mod_rewrite`, you can write a simple rule to automatically redirect users from `http://www.mydomain.com/` to `https://www.mydomain.com/`. If the user requests a specific Web page or CGI script in that URL, it is also maintained by `mod_rewrite`. Once implemented, users who bookmarked or linked to your Web site prior to your deploying SSL will not have to update anything to go to the TLS/SSL encrypted page; their existing `http://` links will automatically convert to `https://`.

Create a Private SSL Key

You need to create a private SSL key for your Apache Web server. You generate the key with a program called OpenSSL using this command:

```
openssl genrsa -des3 -out KEYFILE 2048
```

If you are using Linux, your distribution contains a package called `openssl` that you can install. If you are using Windows, Apache installs OpenSSL in the `bin` directory. To use it, you need to already be in the Apache configuration directory, and then call the OpenSSL executable stored in `..\bin`:

```
cd C:\Program Files\Apache Software  
Foundation\Apache2.2\conf  
..\bin\openssl.exe genrsa -des3 -out KEYFILE  
2048
```

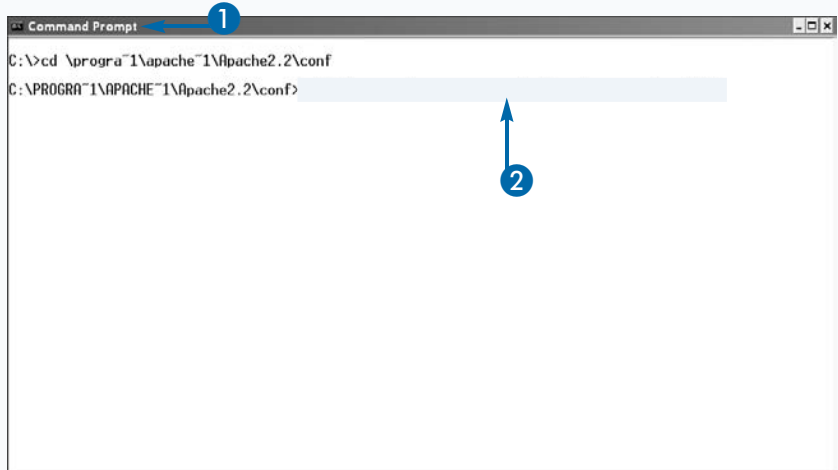
Extra

Because the Apache service needs access to the decrypted key in order to make use of your signed SSL certificate, you need to type in your pass-phrase each time you start Apache. This may not always be practical, especially if you need to reboot your Web server remotely and cannot log in to type it in. You can erase your private key's pass-phrase so Apache can access it unhindered with the following OpenSSL command: You are prompted for a pass-phrase to encrypt the key.

```
openssl rsa -in server.key -out server.  
insecure.key
```

Create a Private SSL Key

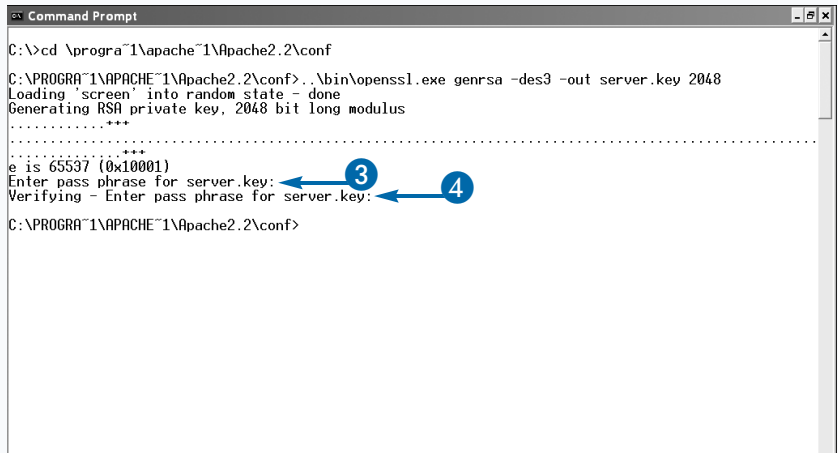
- 1 Open a Terminal window in the Apache configuration directory.
- 2 Type `openssl` (`..\bin\openssl.exe` in Windows) `genrsa -des3 -out KEYFILE 2048` and press Enter.



```
Command Prompt  
C:\>cd \progra~1\apache~1\Apache2.2\conf  
C:\PROGRA~1\APACHE~1\Apache2.2\conf>
```

- 3 Type in a pass-phrase and press Enter.
- 4 Type in the pass-phrase again and press Enter.

OpenSSL generates the private SSL key file.



```
Command Prompt  
C:\>cd \progra~1\apache~1\Apache2.2\conf  
C:\PROGRA~1\APACHE~1\Apache2.2\conf>..\bin\openssl.exe genrsa -des3 -out server.key 2048  
Loading 'screen' into random state - done  
Generating RSA private key, 2048 bit long modulus  
.....+++  
.....+++  
e is 65537 (0x10001)  
Enter pass phrase for server.key:   
Verifying - Enter pass phrase for server.key:   
C:\PROGRA~1\APACHE~1\Apache2.2\conf>
```


Generate an SSL Certificate Signing Request

You need to generate an SSL certificate signing request, or CSR, using OpenSSL and your new private key. The CSR you create is your template for your SSL certificate. This file must be submitted to a certificate authority, or CA, which signs your CSR, producing your final SSL certificate on your behalf.

Each domain name you want to secure with SSL requires its own CSR file. You may re-use the same private key on multiple CSRs:

```
openssl req -new -key KEYFILE -out CSRFILE
```

Remember, if you are using OpenSSL in Windows, you need to be in the Apache configuration directory when you run the OpenSSL executable, which is stored in the `bin` directory:

```
cd C:\Program Files\Apache Software Foundation\Apache2.2\conf
..\bin\openssl.exe req -new -key KEYFILE -out CSRFILE
```

It is generally a good idea to name the file to match the intended domain. For example, `www.mydomain.com` would use `mydomain.csr`. The CSR filename is largely irrelevant, as Apache never uses it directly. However, the signed certificate, the CRT file, typically matches the name of the CSR (for example, `mydomain.crt`). It is the CRT that is referred to by the Apache configuration.

The CSR generation process asks you for some information that is stored in the request file. This information will be used later by a CA to validate that you are who you say you are. The only field that is absolutely required is the common name, or CN. This must match the domain name you will be securing with SSL.

Generate an SSL Certificate Signing Request

1 Type **openssl** (`..\bin\openssl.exe` in Windows) **req -new -key KEYFILE -out CSRFILE** and press Enter.

2 Type the pass-phrase for the private key and press Enter.

3 Type in your two-letter country name and press Enter.

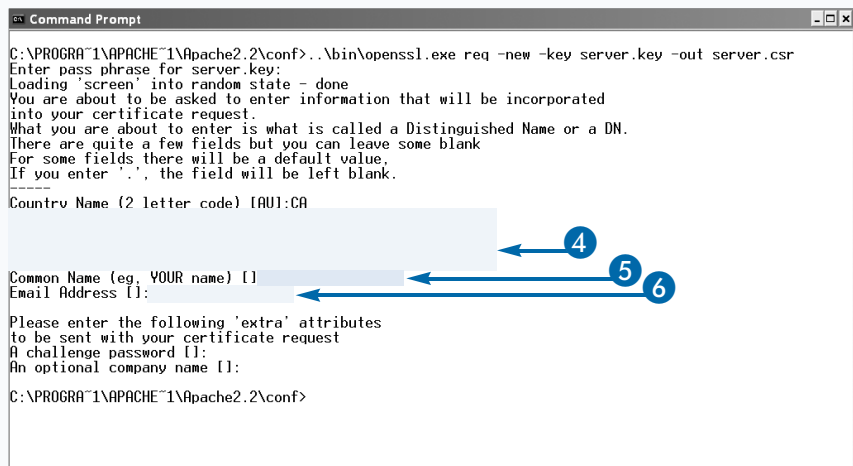
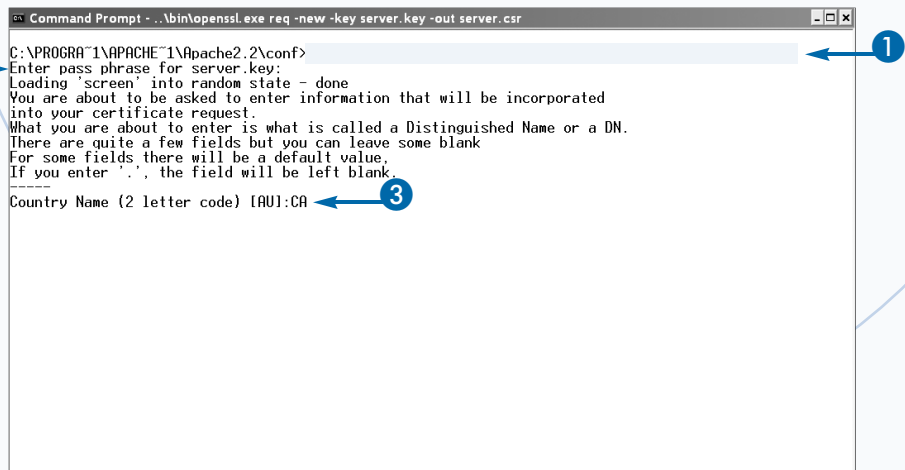
4 Continue entering in your state or province, city, company, and section names.

5 Type your secure domain name, such as **www.mydomain.com**, as the common name and press Enter.

6 Type an e-mail address and press Enter.

7 Press Enter to leave the 'extra' attributes as undefined.

OpenSSL generates the CSR file.



Sign Your Own CSR to Create a Test SSL Certificate

You can sign your own CSR file for testing purposes only. This creates a properly encrypted session between the Web browser and Web server, but any Web browser using the certificate warns the user that the TLS/SSL session is not completely trusted. Self-signing bypasses the step where a CA signs your CSR, so use only self-sign for testing purposes:

```
openssl x509 -signkey KEYFILE -req -in CSRFILE
-out CRTFILE
```

If you are using OpenSSL in Windows, you need to be in the Apache configuration directory when you run the OpenSSL executable, stored in the bin directory:

```
cd C:\Program Files\Apache Software
Foundation\Apache2.2\conf
..\bin\openssl.exe x509 -signkey KEYFILE -req
-in CSRFILE -out CRTFILE
```

Sign Your Own CSR to Create a Test SSL Certificate

- 1 Type **openssl** (**..\bin\openssl.exe** in Windows) **x509 -signkey KEYFILE -req -in CSRFILE -out CRTFILE** and press Enter.
- 2 Type the pass-phrase for the private key.

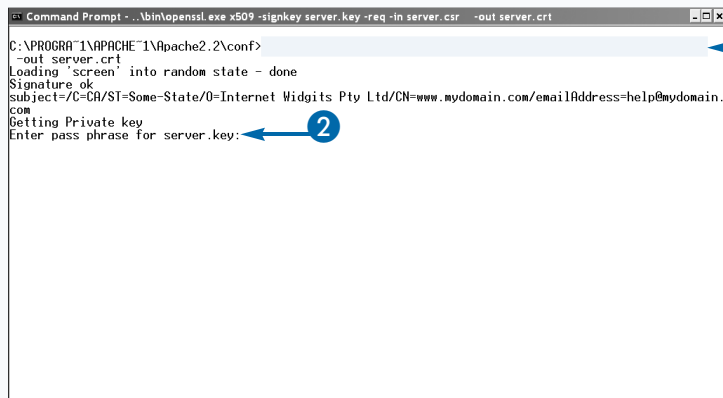
OpenSSL generates the self-signed CRT file.

- 3 Type **openssl** (**..\bin\openssl.exe** in Windows) **x509 -noout -text -in CRTFILE** and press Enter.
- The Terminal window displays the contents of the self-signed CRT file.

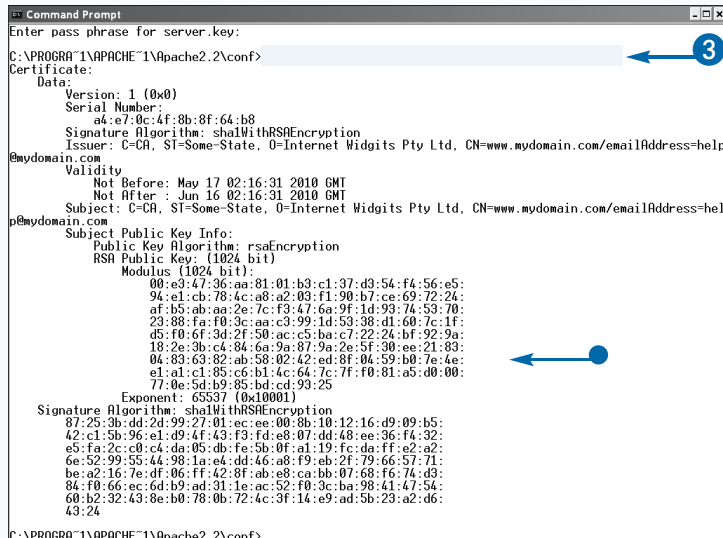
The CRT file's name should match the CSR file, except with a new file extension. For example, *mydomain.csr* should create *mydomain.crt*. Your self-signed CRT file should be stored in the Apache configuration directory.

After a CSR file is signed as a CRT, it is safe to delete the original CSR, as Apache never uses it directly — that is, unless you plan on actually submitting it to a real CA. For more information, see the section, “Submit Your CSR to Be Signed by a Certificate Authority.”

It is actually possible to create your own CA, and then have it sign your CSRs. This is only really practical if you have a large-scale SSL deployment, and can deploy your CA's public certificate onto your users' Web browsers directly. Doing this actually instructs the browser that your CA is to be trusted, as well as any SSL certificate your CA signs.



```
Command Prompt - ..\bin\openssl.exe x509 -signkey server.key -req -in server.csr -out server.crt
C:\PROGRAM FILES\APACHE\Apache2.2\conf>
-out server.crt
Loading screen into random state - done
Signature ok
subject=C=CA, ST=Some-State, O=Internet Wdights Pty Ltd/CN=www.mydomain.com/emailAddress=help@mydomain.com
Getting Private key
Enter pass phrase for server.key:
```



```
Command Prompt
Enter pass phrase for server.key:
C:\PROGRAM FILES\APACHE\Apache2.2\conf>
Certificate:
Data:
  Version: 1 (0x0)
  Serial Number:
    a4:e7:0c:4f:8b:8f:64:b8
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=CA, ST=Some-State, O=Internet Wdights Pty Ltd, CN=www.mydomain.com/emailAddress=help@mydomain.com
  Validity
    Not Before: May 17 02:16:31 2010 GMT
    Not After : Jun 16 02:16:31 2010 GMT
  Subject: C=CA, ST=Some-State, O=Internet Wdights Pty Ltd, CN=www.mydomain.com/emailAddress=help@mydomain.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
    Modulus (1024 bit):
      00:e3:47:96:aa:81:01:b3:c1:37:d3:54:f4:56:e5:
      94:e1:cb:78:4c:a8:a2:03:f1:98:b7:ce:69:72:24:
      af:b5:ab:aa:2e:7c:f3:47:6a:9f:1d:93:74:53:70:
      23:88:fa:f0:3c:aa:c3:99:1d:53:38:d1:60:7c:1f:
      d5:f0:6f:3d:2f:50:ac:c5:bac:f7:22:24:bf:92:9a:
      18:2e:3b:c4:84:6a:9a:87:9a:2e:5f:30:ee:21:83:
      04:83:63:82:ab:58:02:42:ed:8f:04:59:b0:7e:4e:
      el:a1:c1:85:c6:b1:4c:64:7c:7f:f0:81:a5:d0:00:
      77:0e:5d:b9:85:bd:cd:93:25
    Exponent: 65537 (0x10001)
  Signature Algorithm: sha1WithRSAEncryption
    87:25:3b:dd:2d:99:27:01:ec:ee:00:8b:10:12:16:d9:09:b5:
    42:c1:5b:96:e1:d9:4f:43:f3:fd:e8:07:dd:48:ee:36:f4:32:
    e5:fa:2c:c0:c4:da:05:db:fe:5b:0f:a1:19:fc:da:ff:e2:a2:
    6e:52:99:55:44:98:1a:ek:dd:46:a8:f9:eb:2f:79:66:57:71:
    be:a2:16:7e:df:06:ff:42:8f:ab:e8:ca:bb:07:68:f6:74:d3:
    84:f0:66:ec:6d:b9:ad:31:1e:ac:52:f0:3c:ba:98:41:47:54:
    60:b2:32:43:8e:b0:78:0b:72:4c:3f:14:e9:ad:5b:23:a2:d6:
    43:24
C:\PROGRAM FILES\APACHE\Apache2.2\conf>
```

Submit Your CSR to Be Signed by a Certificate Authority

If you plan to deploy SSL, you must get your CSR signed by a certificate authority, or CA. The CA's job is to validate who you say you are by examining your CSR and optionally requesting additional support documentation.

Web browsers are installed with a preconfigured list of CAs, each of which is an independent third party. Because the browser already trusts each CA, SSL dictates that everyone the CA trusts can be implicitly trusted by the end-user.

If you are a business, the CA may request additional documentation, such as a business license certificate, to support your claim as the legitimate trademark owner of an Internet domain name. Deploying SSL on a personal Web site usually means relaxed security checks. The exact process depends on

whom you choose as your CA, and the type of SSL certificate requested. CAs usually charge according to the length of time a certificate is to be valid, as well as for any additional SSL features, such as multiple subdomains, identity verification, and stronger SSL encryption indicators. The annual price ranges from \$30 for a single, unverified, personal domain, to as much as \$599 for a verified corporate domain. Shop around for the best price:

www.godaddy.com/ssl/
www.thawte.com/ssl/
www.verisign.com/ssl/

After you select a CA and submit payment, you need to provide your CSR. Once the CA is satisfied with your request, you receive a signed CRT file that must be applied to your Web site in the Apache configuration using special SSL directives.

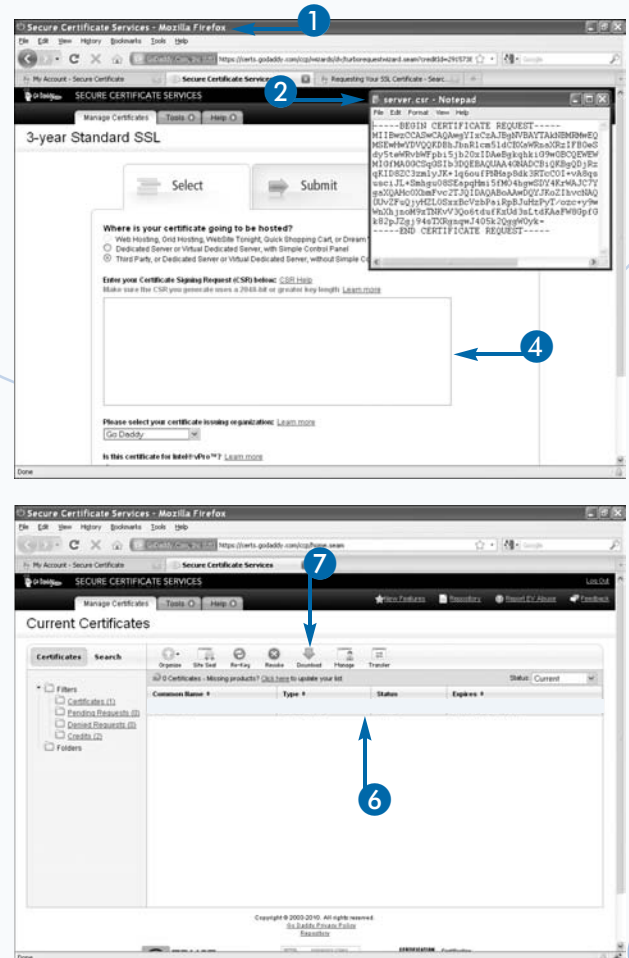
Submit Your CSR to Be Signed by a Certificate Authority

- 1 Open a Web browser and go to a CA Web site.
- Purchase a SSL Certificate plan.
- 2 Open your CSR with a text editor.
- 3 Press **Ctrl+A** to select all text. Press **Ctrl+C** to copy it into your clipboard.
- 4 Press **Ctrl+V** to paste your CSR in to the CA's submission form.
- 5 Complete the form and follow the CA's instructions for processing your request.

Note: If you purchased a corporate SSL certificate, the CA's verification process provides additional instructions that must be followed before you receive your signed CRT file.

The signed CRT file is generated by the CA.

- 6 Locate your certificate on the CA Web site.
 - 7 Download the signed CRT file.
- Save the downloaded CRT file into Apache's configuration directory.



Configure Apache to Use TLS/SSL

Once you have your CSR signed, you are ready to configure Apache to use TLS/SSL on your Web site. Doing so means that all communication with users who access your site with an `https://` URL will be encrypted.

The Apache module that activates TLS/SSL is called `mod_ssl`. Once enabled, it provides additional directives that you can use in the Apache configuration to control how SSL is used on your server. When you install Apache, it provides a preconfigured, example SSL configuration. This example configuration assumes that you only have one domain, and, once the module is activated, you want to allow basic SSL encryption parallel to non-encrypted traffic. The exact process to enable the configuration differs, depending on your server platform.

Configure Apache to Use TLS/SSL

- 1 Open the Apache SSL configuration file in a text editor.

Note: In Windows, this is in the Apache install directory under `conf\extra\httpd-ssl.conf`. In Ubuntu, edit `/etc/apache2/sites-enabled/default-ssl.conf`. In Red Hat, edit `/etc/apache2/conf.d/ssl.conf`.

- 2 Scroll down to `<VirtualHost _default_:443>`.
- 3 Un-comment the `SSLEngine` directive.
- 4 Set its value to `on`.

If you are using Windows, edit the main Apache configuration file and enable `mod_ssl.so` and `httpd-ssl.conf`. Edit `httpd.conf` and un-comment the following directives:

```
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf
```

If you are using Debian or Ubuntu Linux, run the following commands to activate the SSL module and default site configuration:

```
sudo a2enmod ssl
sudo a2ensite default-ssl
```

Regardless of the platform, you need to enable the following SSL directives in the example Apache SSL configuration:

```
SSLEngine on
SSLCertificateFile CRTFILE
SSLCertificateKeyFile KEYFILE
```

```
1
2 This is the Apache server configuration file providing SSL support.
3 It contains the configuration directives to instruct the server how to
4 serve pages over an https connection. For detailing information about these
5 directives see <URL:http://httpd.apache.org/docs/2.2/mod/mod_ssl.html>
6
7 Do NOT simply read the instructions in here without understanding
8 what they do. They're here only as hints or reminders. If you are unsure
9 consult the online docs. You have been warned.
10
11
12
13 Pseudo Random Number Generator (PRNG):
14 Configure one or more sources to seed the PRNG of the SSL library.
15 The seed data should be of good random quality.
16 WARNING! On some platforms /dev/random blocks if not enough entropy
17 is available. This means you then cannot use the /dev/random device
18 because it would lead to very long connection times (as long as
19 it requires to make more entropy available). But usually those
20 platforms additionally provide a /dev/urandom device which doesn't
21 block. So, if available, use this one instead. Read the mod_ssl User
22 Manual for more details.
23
24 #SSLRandomSeed startup file:/dev/random 512
25 #SSLRandomSeed startup file:/dev/urandom 512
26 #SSLRandomSeed connect file:/dev/random 512
27 #SSLRandomSeed connect file:/dev/urandom 512
28
29
30
```

```
69
70
71 # SSL Virtual Host Context
72
73
74 <VirtualHost _default_:443>
75
76 # General setup for the virtual host
77 DocumentRoot "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs"
78 ServerName winxp.vmware.array.org:443
79 ServerAdmin admin@array.org
80 ErrorLog "C:/Program Files/Apache Software Foundation/Apache2.2/logs/error.log"
81 TransferLog "C:/Program Files/Apache Software Foundation/Apache2.2/logs/access.log"
82
83 # SSL Engine Switch:
84 # Enable/Disable SSL for this virtual host.
85 #SSLEngine
86
87 # SSL cipher Suite:
88 # List the ciphers that the client is permitted to negotiate.
89 # See the mod_ssl documentation for a complete list.
90 #SSLCipherSuite ALL:!ADH:!EXP:RSA:+HIGH:+MEDIUM:+LOW:+SSLV2:+EXP:+NULL
91
92 # SERVER Certificate:
93 # Point SSLCertificateFile at a PEM encoded certificate. If
94 # the certificate is encrypted, then you will be prompted for a
95 # pass phrase. Note that a kill -HUP will prompt again. Keep
96 # in mind that if you have both an RSA and a DSA certificate you
97 # can configure both in parallel (to also allow the use of DSA
98 # ciphers, etc.)
```

- 5 Locate the directive `SSLCertificateFile`.
- 6 Set its value to the full path to your CRT file.
- 7 Locate the directive `SSLCertificateKeyFile`.
- 8 Set its value to the full path to your KEY file.
- 9 Save the Apache SSL configuration file.
- 10 Restart the Apache Web server.

Note: For platform-specific instructions on starting and stopping Apache on Windows, see Chapter 4. For platform-specific instructions on starting and stopping Apache on Linux, see Chapter 5.

Apache restarts.

```

90 SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
91
92 # Server Certificate:
93 # Point SSLCertificateFile at a PEM encoded certificate. If
94 # the certificate is encrypted, then you will be prompted for a
95 # pass phrase. Note that a kill -HUP will prompt again. Keep
96 # in mind that if you have both an RSA and a DSA certificate you
97 # can configure both in parallel (to also allow the use of DSA
98 # ciphers, etc.)
99
100 #
101 # Server Private Key:
102 # If the key is not combined with the certificate, use this
103 # directive to point at the key file. Keep in mind that if
104 # you've both a RSA and a DSA private key you can configure
105 # both in parallel (to also allow the use of DSA ciphers, etc.)
106
107 #
108 # Server Certificate Chain:
109 # Point SSLCertificateChainFile at a file containing the
110 # concatenation of PEM encoded CA certificates which form the
111 # certificate chain for the server certificate. Alternatively
112 # the referenced file can be the same as SSLCertificateFile
113 # when the CA certificates are directly appended to the server
114 # certificate for convenience.
115 #SSLCertificateChainFile "/C:/Program Files/Apache Software Foundation/Apache2.2/conf/sslrver-ca.crt"
116

```



Apply It

If you want to enable SSL on multiple domains, you must apply these SSL directives into each domain's `<VirtualHost domain:443>` configuration group. This customizes which directives are enabled for each domain, allowing each one to have unique key and certificate configuration.

You may want to copy all the default SSL directives within the example `<VirtualHost _default_:443>` group into each domain's configuration group. However, at minimum, you only need three SSL-specific directives — `SSLEngine`, `SSLCertificateFile`, and `SSLCertificateKeyFile` — alongside the normal virtual host directives.

TYPE THIS

```

<VirtualHost www.newdomain.com:443>
    DocumentRoot PATH
    ServerName www.newdomain.com:443
    SSLEngine on
    SSLCertificateFile CRTFILE
    SSLCertificateKeyFile KEYFILE
</VirtualHost>

```

RESULTS

The URL `https://www.newdomain.com/` now uses its own SSL key and certificate.

For more information about how to use `<VirtualHost>`, go to <http://httpd.apache.org/docs/2.2/mod/core.html#virtualhost>.

continued

Configure Apache to Use TLS/SSL (continued)

Different Web browsers respond differently to the TLS/SSL handshake process. A problem occurs when visiting a Web site that uses an SSL certificate that has expired, been signed by an un-trusted CA, assigned to a different domain name, or improperly configured in Apache.

If the connection process succeeds, Firefox and Internet Explorer display a lock icon in the bottom-right corner of the browser, and Google Chrome in the URL bar. Clicking on the lock icon brings up some technical details about the level of encryption that is established. If the connection fails, a warning message appears and the user has the option to continue to connect using TLS/SSL anyway, or to cancel loading the Web page. Internet Explorer displays a Security Alert popup that summarizes the problem, but provides an easy way to continue on to

the site. Google Chrome changes the browser background to red with a warning message, but still allows for an easy way to continue with the click of a button. Firefox displays a less threatening message, but the steps needed to proceed are significantly more complicated. Firefox still wants users to have the option to continue, but emphasizes what is going on by requiring the user to make multiple clicks to allow an exception, thus lessening the chances of inattentive users continuing accidentally.

This behavior of Firefox can be annoying, especially for Web developers trying to debug SSL. It does provide the option to permanently store an exception rule so that specific SSL warnings will be silently ignored in the future. Unfortunately, a permanent exception rule makes it more difficult to identify when the original SSL problem has been legitimately fixed.

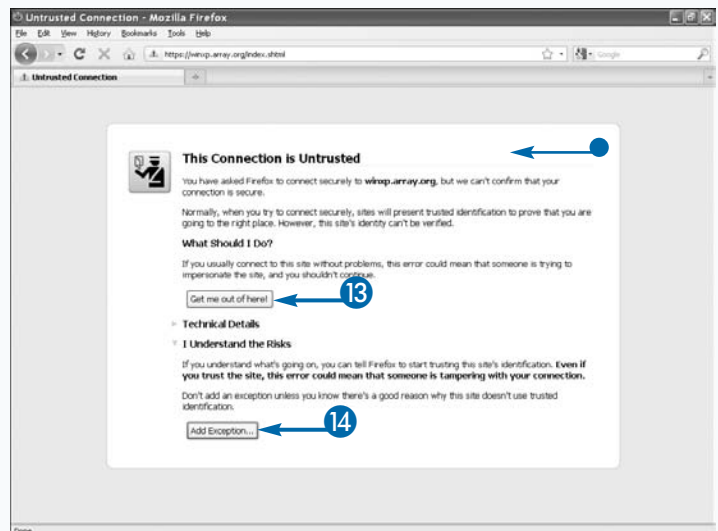
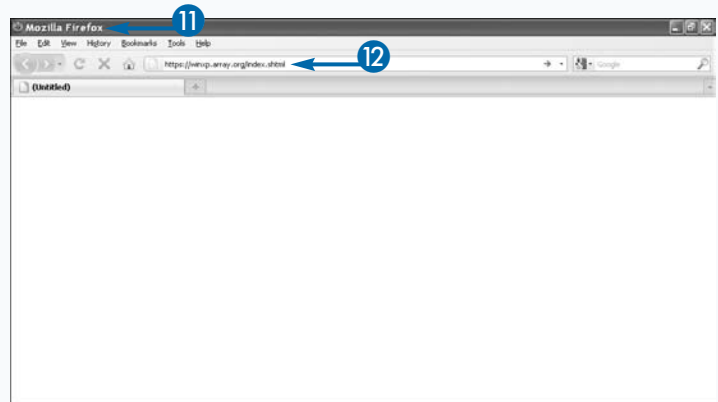
Configure Apache to Use SSL (continued)

- 11 Open a Web browser.
- 12 Type **https://www.mydomain.com** and press Enter.

The SSL/TLS handshake process begins between the Web browser and Apache.

If a problem is detected, the Web browser intervenes.

- The warning message in Mozilla Firefox.
- 13 Click I Understand the Risks.
 - 14 Click Add Exception.



- The Add Security Exception dialog box appears.

15 Click Get Certificate.

- The problems with the current certificate appear.

Note: Click View to see the information about the current certificate. This matches the information you provided in the section, “Generate an SSL Certificate Signing Request,” earlier in this chapter.

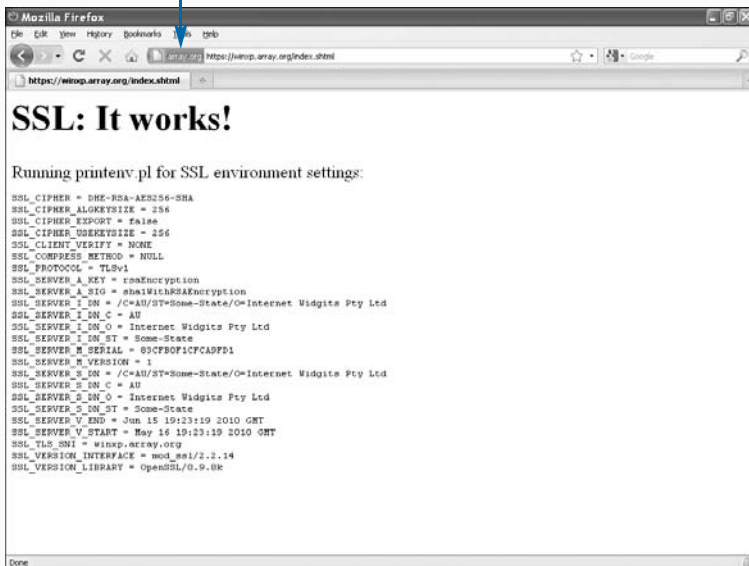
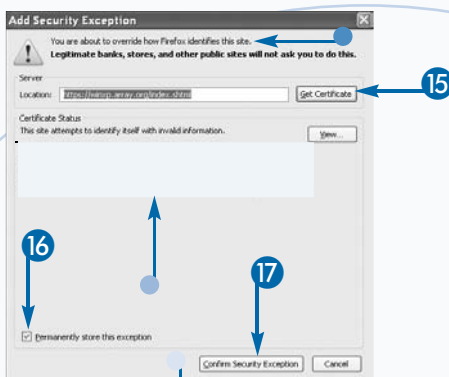
16 Click the Permanently store this exception check box.

17 Click Confirm Security Exception.

The browser loads the TLS/SSL encrypted Web page.

- Firefox highlights the encrypted domain name near the URL.
- Firefox displays a clickable lock icon.

Note: In Firefox, to delete a permanently stored exception rule, click Tools → Options → Advanced → Encryption → View Certificates → Servers. Select an exception rule and click Delete.



Apply It

It is possible to easily force all traffic on your Web site to use an encrypted TLS/SSL connection. You do this by creating a new <VirtualHost domain:80> group running a simple rewrite rule on the non-encrypted port, alongside the real <VirtualHost domain:443> group with the actual Web site directives.

For this to work, you need to make sure the `mod_rewrite.so` module is active in Apache. You can follow the same activation procedure that you used to enable `mod_ssl.so` earlier in this section, or manually specify the `LoadModule` directive with the path to `mod_rewrite.so`.

TYPE THIS

```
LoadModule rewrite_module modules/mod_rewrite.so
<VirtualHost www.mydomain.com:80>
    RewriteEngine on
    RewriteRule (.*)
    https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>
```



RESULTS

All users requesting the non-encrypted URL `http://www.mydomain.com/` are automatically redirected to `https://www.mydomain.com/`. The advantage of this method is that any legacy bookmarks or external links going to a specific non-encrypted HTML or CGI address are silently redirected to the encrypted address.

Understanding Security in Perl CGI Development

No programming language is completely secure. Regardless of the effort made by the language's designers to make it so, it is impossible to guarantee 100-percent security if its users (that is, you) do not follow best practices and common sense in a program's design. Even if the latest model car has the most advanced safety features, it cannot stop you from driving on the wrong side of the road, or from getting into an accident.

Naturally, if your program displays a particular security flaw, it is possible that it was not caused directly by your design. Sure, you may discover a way to manipulate your own code to eliminate the problem, or perhaps an upgrade or patch already exists online, but the point is that you are ultimately responsible for keeping your Web site, server, and the user's data safe.

Even before such a security flaw can be exploited, there are steps you can take now to minimize your exposure so that an attacker cannot leverage it. You do this by disabling any unnecessary system access and protocols, error-correcting data entered by the end-user, and monitoring the system's log files. These recommendations are not a complete list, but they are a good start.

The best way to identify any potential security problem is to exploit it yourself. Put yourself in the shoes of a potential attacker, and ask, "What on my Web site is of any value? What can I not afford to lose?"

Only by devising a scenario and gaining insight from each participant — including the end-user, their Web browser, the Web server, and your Perl code — can you find the weak link, construct a plan, and correct the situation. Ask yourself, "If I purposely do something unexpected and uncommon, something a normal user is never likely to do, how will my program respond?"

Controlled CGI Execution

Even if you are the only active user on your Apache Web server, and you already have root or administrator access on the system, you should only allow CGI execution conservatively and where appropriate.

Limiting CGI and SSI

Remember, CGI is a direct link from remote URL to local program execution. If a malicious CGI script is installed in a CGI-enabled directory, Apache will readily execute it. You must limit CGI execution to only pre-approved, limited-access directories, and restrict permissions so that unauthorized users cannot read or write files on your Web server.

You can easily restrict SSI from running programs, CGI or native, and maintain its ability to read and import other HTML files. Unless you have a specific requirement for an SHTML page to import actual program output, you should only enable SSI with the Apache directive `IncludesNoExec`.

Executing CGI as the User

Your Web server has built-in controls for native program execution, provided by the operating system. When a regular user runs any program, unauthorized system calls are automatically restricted. Apache follows this convention by executing all CGI scripts as the user who owns the script.

In other words, if your CGI script always runs as a regular user, and it does something damaging, the scope of impact on other users and the core OS is automatically limited.

One Script, Multiple Pages, Specific Workflow

A CGI script that implements multiple HTML forms, such as a shopping-cart checkout program, requires the user to progress through each page, but stay in the same URL. The script may implement a specific workflow where the user must progress from one screen to the other in order. For each page load, your script must decide where the user is in the process, and display the appropriate page. If in the middle of the process the user does something unusual, such as clicking Refresh, or setting a bookmark to return later, the CGI must correctly filter out duplicate or incomplete form data correctly. Otherwise, the user could end up in an unexpected session state with unforeseen security implications.

Initial CGI Load

Obviously, when a complex CGI script is first loaded, the first page appears. The challenge here is that the CGI protocol is not session-aware, so it is up to the script to identify if the user is new to the process, or further along. An HTTP cookie that stores a unique session identifier is a convenient way of identifying multiple users who are simultaneously using the CGI script.

The session identifier's value needs to be unique for every user, and impossible to guess. Otherwise, another user's session could be hijacked, revealing personal information to the attacker about that user's session and activity. A good identifier uses two or three values independent of one another, concatenated into a single string. For instance, appending a timestamp, a process ID, and a random number produces a session identifier value that is guaranteed to be unique and virtually impossible to guess:

```
my $sessionId = $$ . time . rand( 5 );
```

The session identifier is a static value; your code will need to correctly identify new users from returning users and use the previously assigned session identifier from the cookie, if found. For an example of how to do this, see Chapter 15.

Progressing Through the CGI's Screens

The idea behind a user session is an unknown concept to the Web browser and Web server. It is your Perl CGI script's responsibility to follow the user's activity from page to page, and to make sense of the information that it receives.

As each page is submitted, the CGI must decide if the values collected satisfy the previous form's fields. Were the first and last name fields populated? Is the phone number formatted like a phone number? Does the e-mail address field contain an "@" and a domain?

If something is not properly submitted, the user should not be allowed to continue. The last form must be displayed again along with an error message. It is a good practice to pre-populate the fields that were submitted correctly, so that the user only needs to correct the invalid fields and resubmit the form.

If everything checks out, the CGI appends the new information into the database, linked to the session identifier, and allows the user to proceed to the next page in the workflow.

The advantage of this whole process is that the URL never changes. By limiting the user to simply submitting form data, the CGI maintains control over the workflow, and decides what to display based upon their status. You effectively make it impossible for the user to drift outside of the CGI's hard-coded process. Even if the user manages to change or delete their session cookie, the worst that can happen is that their previous session is left stagnant and they must start over.

Limit CGI Access in Apache

You can limit CGI access in Apache to only pre-approved users and directories that require it. CGI is inherently powerful; by restricting these avenues of execution, it will help you to minimize the potential for unauthorized users to install and execute unauthorized CGI scripts on your server.

It is always a good practice to limit which users have write access to CGI directories, regardless of whether you are the only person who can access the Web server. This restriction makes it very difficult for an unauthorized user account, one that has been compromised and accessed by an attacker, to install a malicious CGI program. In fact, you should also apply this simple rule to all HTML directories, thus limiting the chances of irrevocably altering your entire Web site.

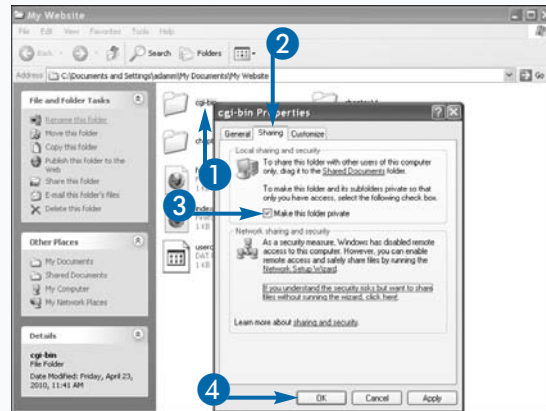
An additional layer of security is applied to CGI scripts by a module called suEXEC. Because Apache runs as a system-level user, suEXEC allows Apache to switch its *effective user* to the CGI script's owner account when executed. This module limits the potential impact of compromised CGI scripts because non-privileged users have far fewer opportunities to cause significant damage to the core system functionality. This feature is only available on Unix Apache installs; no equivalent feature exists for Windows.

The methods described in this book are not guaranteed to make your Web site unhackable, but they do block common avenues that an attacker would attempt to leverage, thus making it more difficult for you to be directly affected.

Limit CGI Access in Apache

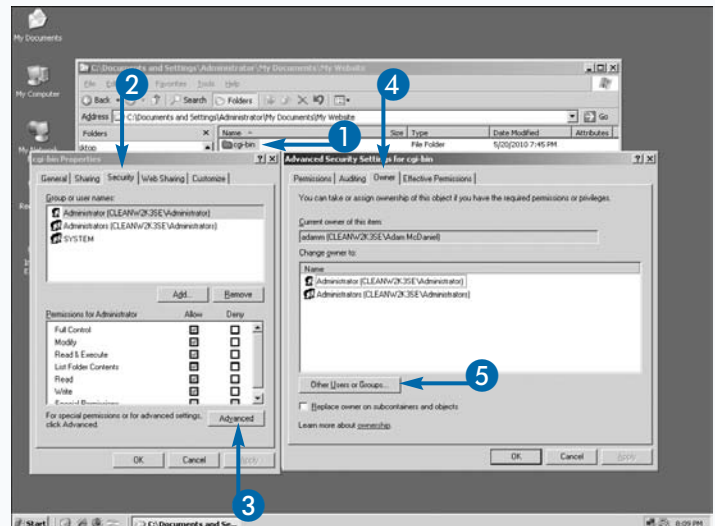
Windows XP Only

- 1 Open the properties of your cgi-bin folder.
- 2 Click Sharing.
- 3 Click the Make this folder private check box.
- 4 Click OK.

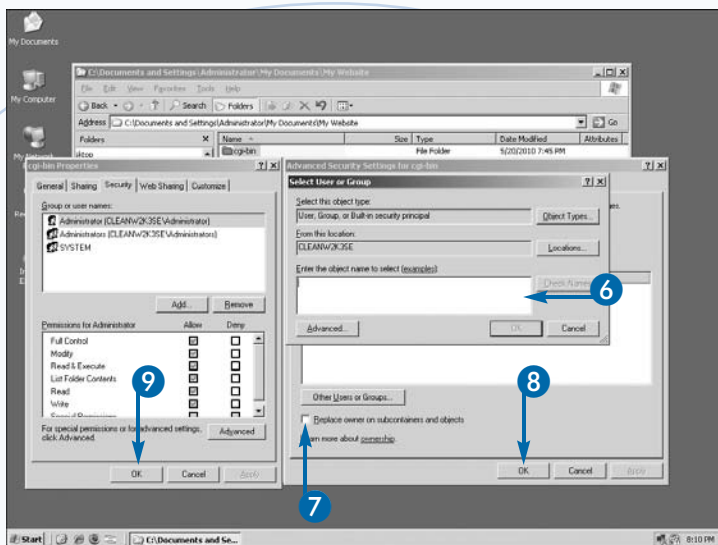


Windows Server Only

- 1 Open the properties of your cgi-bin folder.
- 2 Click Security.
- 3 Click Advanced.
- 4 In the Advanced Security Settings dialog box, click Owner.
- 5 Click Other Users or Groups.

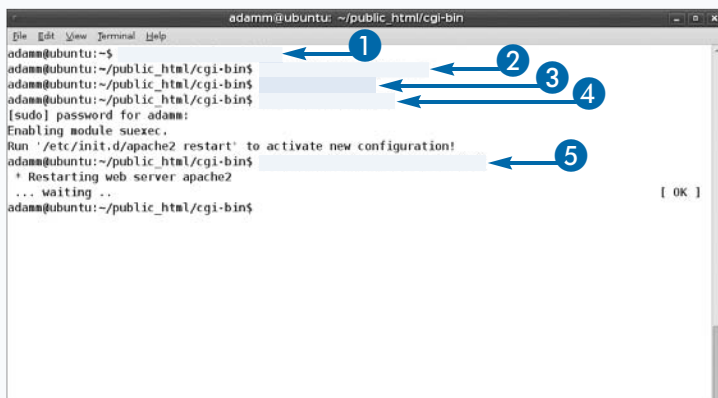


- 6 In the Select User or Group dialog box, type your username and click OK.
- 7 In the Advanced Security Settings dialog box, enable the check box for Replace owner on subcontainers and objects.
- 8 Click OK.
- 9 In the Properties dialog box, click OK.



Ubuntu Linux Only

- 1 Open a Terminal window in your `cgi-bin` directory.
- 2 Type `chown -R USER.USER . *` and press Enter.
- 3 Type `chmod -R 755 . *` and press Enter.
- 4 Type `sudo a2enmod suexec` and press Enter.
- 5 Restart Apache.



Note: The `suEXEC` module is unavailable for Windows.

Extra

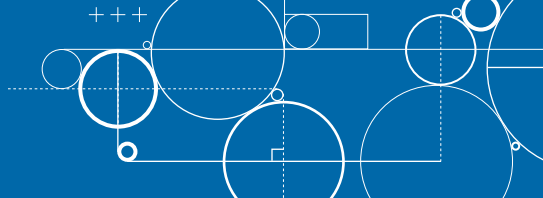
Note that Fedora Linux ships with `suEXEC` preconfigured. On other Unix distributions, you may need to manually enable the appropriate `LoadModule` directive:

```
LoadModule suexec_module path/mod_suexec.so
```

The examples here demonstrate one possible way to *harden*, or lock down, your `cgi-bin` directory, which limits the chances of someone else installing their own CGI script here. However, this is making the assumption that the core operating system, and even the physical server, are both sufficiently secured. If an attacker gains access to your account, the root account, or even physical access to the room that holds your server, it does not matter what you do to harden the Apache configuration. If the attacker can impersonate you, then Apache and the operating system will readily oblige their commands.

There have been volumes of books written about server hardening, such as *Microsoft SQL Server 2008 Management and Administration* (Wiley, 2009), *Professional Windows Desktop and Server Hardening (Programmer to Programmer)* (Wiley, 2006), *Ubuntu Linux Bible: Featuring Ubuntu 10.04 LTS* (Wiley, 2010), *Linux Bible 2010 Edition: Boot Up to Ubuntu, Fedora, KNOPPIX, Debian, openSUSE, and 13 Other Distributions* (Wiley, 2010), and *Linux Command Line and Shell Scripting Bible* (Wiley, 2008).

Identify Unusual Activity on Your Web Site



You need to watch for unusual activity on your Web site. Users will always find creative and unexpected ways to interact with your CGI scripts. Attackers often use programs that automatically probe Web sites for holes and known weaknesses. Even before your Web site becomes popular, it is extremely important to monitor the activity on your server. By catching and identifying unusual activity, you can apply code into your CGI scripts to circumvent it.

Unusual activity is a rather vague concept. How can you watch for and circumvent it if you do not know exactly what it is? Simply put, this is any Web site activity that you did not plan for.

One way to practice identifying it is to do something unusual on your own Web site. Try typing in a bunch of garbage characters into one input field. Try typing in HTML code in another. Try changing your own session identifier cookie midway through a complex CGI process. These three examples are the kinds of unusual activity that you need to learn how to identify, trap, and counter.

Monitor the Apache Activity and Error Logs

As discussed in Chapter 10, all hits to your Web site are logged to the `access.log` file, and any errors are summarized into the `error.log` file. The error reporting is rather limited, but you can retrieve information such as when an error occurred, the user's IP address, and the URL they accessed.

Install an Apache Log Scanner

A log scanner is a program designed to monitor large amounts of logged activity for you, and to either send you a summary of anything interesting, or simply take appropriate action. If a single user is initiating an unusual amount of errors, this could be a sign of an attacker probing your Web site for weaknesses. A log scanner can isolate that user and apply a rule that bans them from your Web site.

fail2ban

fail2ban is a Python script designed to monitor Apache and SSH authentication requests by scanning system log files. You could easily apply this to your Web site if you implement Apache User Authentication, as discussed in the first half of Chapter 15.

When fail2ban identifies a potential attacker, that person's IP address is added to the local system's firewall table, banning them from being able to connect to your Web server. Because fail2ban uses the program iptables to do this, only Linux is supported.

You can find more information on the fail2ban Web site, at www.fail2ban.org.

ScanErrLog

ScanErrLog is also a Python script, but it is designed for on-demand error log checking, and focusing on the actual error messages, not the users who cause them. It produces an HTML, PDF, or XML summary of what it finds, allowing you to examine which component of your Web site is failing. It also counts how many times a particular error has occurred, which is useful for comparing these results over time.

This program is more focused on identifying problems so dynamic Web site developers can address their code and minimize anything that causes an error. For more information, go to www.librelogiciel.com/software/ScanErrLog/action_Presentation.

Generate Your Own Activity Log

You can easily generate a custom activity log of any event handled by Perl CGI scripts. If you choose to do this, you can create a subroutine that identifies the file, and writes key identifying data such as a timestamp and the script's filename, along with a custom text message:

```
sub logger {
    my $msg = shift;
    my $time = localtime;
    open( OUTPUT, ">> FILE.log" );
    printf OUTPUT "%s - %s: %s\n", $time, $0,
    $msg;
    close( OUTPUT );
}
```

Each time you call this subroutine, the timestamp, filename, and message text are appended to a log file.

You can use the Perl module `Log::Agent` for custom logging. Available on CPAN, `Log::Agent` provides an abstract way to add logging into individual modules, while `Log::Agent::Logger` in your actual Perl CGI scripts channels the activity to an output file. You can configure the module `Log::Agent::Rotate` to automatically rotate a growing log file and archive it. It can even clear archived logs after a number of days to help you manage disk space.

Monitor Submitted Form Data Carefully

When a user submits data in an HTML form, anything can be sent to your Web server, anticipated or not. You must code your Perl CGI scripts to handle anything the server receives. There are some simple techniques in Perl to sanitize and validate all CGI data that is submitted.

Apply Sanitizing Rules

All data that users submit should be sanitized for inappropriate content. Usually this means removing unusual characters, such as excessive spacing and non-alphanumeric symbols, and stripping out anything that looks like an HTML tag. You may also want to automatically filter inappropriate words, or even implement a moderator-style workflow.

For example, if you allow regular users to write comments on your Web pages or blog posts, be very careful about making those comments appear online immediately. You can use Perl to do some of the sanitization work for you, but you may also need to manually review and approve it.

Search-and-replace statements using a regular-expression pattern are the best way to implement automatic data sanitization.

Apply Validation Checks

The validation check happens after input data has passed sanitization. This ensures that the user's input for a particular field matches its predefined format. A validation check may verify that a phone number has the correct number of digits and dashes, that an e-mail address is genuine, or simply that a required field was not ignored.

You can implement a validation check with a conditional statement, followed by a regular-expression pattern match. You then need to decide what to do if the check identifies a failure. You need to communicate back to the user, indicating what field has failed, how it failed, and that the user must try submitting the form again.

Reviewing Collected Data

Always perform routine reviews of the data you have accepted into your database. Even though this is all content that has passed your sanitization and validation checks, you may find an anomaly that should not be there. It may be quickest to edit the database directly and manually fix the offending field, but do not forget to update your CGI code to stop this problem from happening again.

Remember, your sanitization and validation rules should evolve as more users visit your Web site. They will continually find more unusual text to enter, which you must catch; otherwise, you risk affecting your core data integrity, and potentially your Web site's reputation.

Sanitize User Content in Perl CGI

You can sanitize all user content submitted to your Web site with Perl search-and-replace regular expressions. Sanitizing a field keeps the response, but filters out garbage and common user mistakes.

```
VAR =~ s/pattern/replace/flags;
```

This statement searches *VAR* for the regular expression *pattern*. Any matches are changed to *replace*. Optional *flags* alter how the whole matching process works. For example, you can sanitize all phone numbers by removing all characters but numbers and dashes:

```
$phone =~ s/[^\d\-]//g;
```

There are some basic rules you can use to sanitize every submitted field. Removing all excessive spacing at the beginning, end, and middle of text-based variables is a very easy way to ensure that user-submitted data uses clean spacing:

Sanitize User Content in Perl CGI

- 1 Type `my @fields = (field1, field2, ...);`.
- 2 Type `my $params = {};`.
- 3 Type `foreach (@fields) {`.
- 4 Type `$params->{ $_ } = $cgi->param($_);`.
- 5 Type `}`.

Note: This `foreach` loop cycles through all HTML fields populated by the user.

- 6 Type `$params->{ $_ } =~ s/^\s+//;`.
- 7 Type `$params->{ $_ } =~ s/\s+$//;`.
- 8 Type `$params->{ $_ } =~ s/\s+/ /g;`.

Note: `\s+` is a whitespace pattern, but a literal space is used as the replacement. This way, even a tab or new line will be converted into a space.

```
+++  
$field =~ s/^\s+//;  
$field =~ s/\s+$//;  
$field =~ s/\s+/ /g;
```

The first statement removes all whitespace from the start of `$field`. The second statement removes it from the end. The last statement ensures that multiple spaces anywhere in the middle are replaced with a single space. Once implemented, the submitted text “ This is my data ” is sanitized to “This is my data”.

You should also filter out anything that looks like an HTML tag. This completely eliminates the potential for an HTML-injection attack by someone submitting a cleverly crafted `<script>` or `<object>` tag as form input:

```
$field =~ s/<.*?>/g;
```

Sanitizing user content may seem unnecessary, but you will encounter users that warrant it.

```
1 #!C:\Perl\bin\perl.exe  
2  
3 use CGI;  
4 use HTML::Template;  
5  
6 my $cgi = new CGI;  
7 my $tmpl = HTML::Template->new( filename => 'process-form.tmpl',  
8   die_on_bad_params => 0 );  
9  
10 if ( $cgi->param( "process" ) ) {  
11   my @fields = ( 'firstname', 'lastname', 'email', 'phone' );  
12   my $params = {};  
13  
14   foreach ( @fields ) {  
15     $params->{ $_ } = $cgi->param( $_ );  
16  
17   }  
18  
19  
20   my $values = {};  
21   foreach ( @fields ) {  
22     push( @{$values}, {  
23       name => $_,  
24       value => $params->{ $_ },  
25     } );  
26   }  
27   $tmpl->param( valueList => $values );  
28 }  
29  
30 print $cgi->header();
```

```
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
20
```

9 Type `$params->{ $ _ } = ~ s/<.*?>/g;`

10 Type `$params->{ $ _ } = ~ s/[^\w\s_\-\.\&@]/g;`

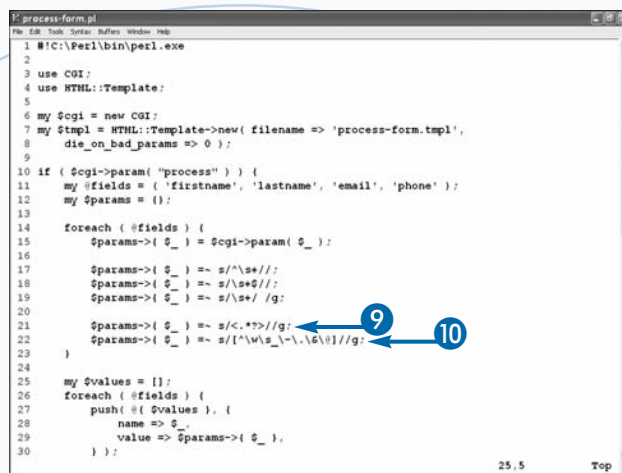
Note: This filter removes all characters that are not alphanumeric or a space, underscore, dash, period, ampersand, or @ sign for all CGI input values. This is very useful as a generic catch-all filter.

11 Type `$params->{ field } =~ s/s/-/g;`

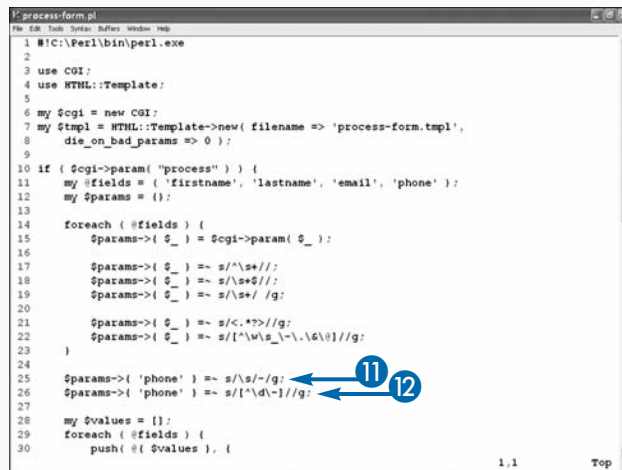
Note: This statement converts all spaces to dashes, but only for a specific field.

12 Type `$params->{ field } =~ s/[^\d-]/g;`

Note: This statement is a regular expression that strips everything else, excluding numbers and dashes, from field.



```
1 #!C:\Perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new( filename => 'process-form.tmpl',
8   die_on_bad_params => 0 );
9
10 if ( $cgi->param( "process" ) ) {
11   my @fields = ( 'firstname', 'lastname', 'email', 'phone' );
12   my $params = {};
13
14   foreach ( @fields ) {
15     $params->{ $_ } = $cgi->param( $_ );
16
17     $params->{ $_ } =~ s/^se//;
18     $params->{ $_ } =~ s/\s$/;
19     $params->{ $_ } =~ s/se/ /g;
20
21     $params->{ $_ } =~ s/<.*?>/g;
22     $params->{ $_ } =~ s/[^\w\s_\-\.\&@]/g;
23   }
24
25   my $values = {};
26   foreach ( @fields ) {
27     push( @{ $values }, {
28       name => $_,
29       value => $params->{ $_ },
30     } );
31   }
32 }
```



```
1 #!C:\Perl\bin\perl.exe
2
3 use CGI;
4 use HTML::Template;
5
6 my $cgi = new CGI;
7 my $tmpl = HTML::Template->new( filename => 'process-form.tmpl',
8   die_on_bad_params => 0 );
9
10 if ( $cgi->param( "process" ) ) {
11   my @fields = ( 'firstname', 'lastname', 'email', 'phone' );
12   my $params = {};
13
14   foreach ( @fields ) {
15     $params->{ $_ } = $cgi->param( $_ );
16
17     $params->{ $_ } =~ s/^se//;
18     $params->{ $_ } =~ s/\s$/;
19     $params->{ $_ } =~ s/se/ /g;
20
21     $params->{ $_ } =~ s/<.*?>/g;
22     $params->{ $_ } =~ s/[^\w\s_\-\.\&@]/g;
23   }
24
25   $params->{ 'phone' } =~ s/s/-/g;
26   $params->{ 'phone' } =~ s/[^\d-]/g;
27
28   my $values = {};
29   foreach ( @fields ) {
30     push( @{ $values }, {
31       name => $_,
32       value => $params->{ $_ },
33     } );
34   }
35 }
```

Apply It

If you want to keep the literal HTML code, but not have it interpreted by a browser, you can convert the angle brackets in the tags into HTML character entities.

TYPE THIS

```
$param->{ $ _ } =~ s/</&lt;/g;
$params->{ $ _ } =~ s/>/&gt;/g;
```



RESULTS

The HTML tags are converted into a string that is no longer interpreted by the browser as HTML. For example, “text” becomes “text”.

This has the advantage of maintaining the essence of the original HTML code, only now when it displays in the browser, it's like only seeing the literal HTML source.

For more information about regular-expression pattern matching, see the PerlDoc page `perlre`. For information about search-and-replace operations, see the page `perlop`. You can also find a summary of regular Perl expression elements and formats in Appendix A.

Validate User Content in Perl CGI

Validating user content implies testing individual fields for correctly structured values, after they have passed sanitization, and rejecting the values outright if they fail. As in the section, “Sanitize User Content in Perl CGI,” you are using regular expressions, but here you are just matching a pattern, not replacing it.

Matching a phone number is not overly difficult, assuming you do not need to worry about international phone numbers. The United States, Canada, and many Caribbean countries use a standardized “###-###-####” format. Using a conditional test and regular expression pattern, you can check if the user’s input matches this format:

```
if ( $phone =~ /^d{3}-d{3}-d{4}$/ ) {  
    # This looks like a phone number!  
}
```

Validate User Content in Perl CGI

- 1 Type if (! \$params->{ field }) {

Note: Placing this check within the foreach(@fields) block results in all fields being required.

- 2 Insert code to raise the error on field if it is blank.
- 3 Type }.

- 4 Type if (\$fields->{ field } !~ /pattern/) {.

Note: In regular-expression conditional tests, the operator “=~” returns true for a positive pattern match, and “!~” returns true for a negative match.

- 5 Insert code to raise the error on field if it does not match the regular expression pattern.
- 6 Type }.

To validate an e-mail address, you can use the third-party Perl module Mail::RFC822::Address. This module uses regular expressions that are preconfigured to match standard e-mail addresses. Everything is accessible from a single function called valid:

```
use Mail::RFC822::Address 'valid';  
if ( valid( $email ) ) {  
    # This looks like an email address!  
}
```

Sure enough, random characters can still be made to look like an e-mail address or phone number, provided that they follow the appropriate syntax. The only way to be completely sure that an e-mail address is genuine is to send it a message with instructions or a special URL back on your Web site. This form of validation goes beyond regular-expression pattern matching, but it serves the same purpose.

```
process-form.pl  
File Edit Tools System Buffers Window Help  
10 if ( $cgi->param( "process" ) ) {  
11     my @fields = ( 'firstname', 'lastname', 'email', 'phone' );  
12     my $params = {};  
13  
14     foreach ( @fields ) {  
15         $params->{ $_ } = $cgi->param( $_ );  
16  
17         $params->{ $_ } =~ s/^\s+//;  
18         $params->{ $_ } =~ s/\s+$//;  
19         $params->{ $_ } =~ s/</>/ /g;  
20  
21         $params->{ $_ } =~ s/<.*>//g;  
22         $params->{ $_ } =~ s/[^\w\s_\-\.@{}]/ /g;  
23     }  
24  
25     $params->{ 'phone' } =~ s/\s+/-/g;  
26     $params->{ 'phone' } =~ s/[^\d-]/ /g;  
27  
28     foreach ( @fields ) {  
29         if ( ! $params->{ $_ } ) {  
30  
31  
32     }  
33  
34     my $values = {};  
35     foreach ( @fields ) {  
36         push( @{$values}, {  
37             name => $_,  
38             value => $params->{ $_ },  
39  
33,6 56%
```

```
process-form.pl  
File Edit Tools System Buffers Window Help  
22 $params->{ $_ } =~ s/[^\w\s_\-\.@{}]/ /g;  
23 }  
24  
25 $params->{ 'phone' } =~ s/\s+/-/g;  
26 $params->{ 'phone' } =~ s/[^\d-]/ /g;  
27  
28 foreach ( @fields ) {  
29     if ( ! $params->{ $_ } ) {  
30         # Raise an error to the user.  
31         $errors->{ $_ } = "$_ is missing";  
32     }  
33  
34  
35     if ( $fields->{ 'phone' } !~ /^d{3}-d{3}-d{4}$/ ) {  
36         $errors->{ 'phone' } = "Phone number incorrectly formatted";  
37     }  
38  
39     my $values = {};  
40     foreach ( @fields ) {  
41         push( @{$values}, {  
42             name => $_,  
43             value => $params->{ $_ },  
44         } );  
45     }  
46     $tmpl->param( valueList => $values );  
47 }  
48  
49 print $cgi->header();  
50 print $tmpl->output;  
45,6 Bot
```

- 7 Display any errors raised during this process to the user by using a template parameter.

```

1 process-form.pl
2
3 $params->{ '$_' } =~ s/{^\w{s}_\-\.\d{0}}//g;
4
5 $params->{ 'phone' } =~ s/{s/-/g;
6 $params->{ 'phone' } =~ s/{^\d{1}}//g;
7
8 foreach ( @fields ) {
9     if ( ! $params->{ '$_' } ) {
10         # Raise an error to the user.
11         $errors->{ '$_' } = "$_ is missing";
12     }
13 }
14
15 if ( $fields->{ 'phone' } !~ /\d{3}-\d{3}-\d{4}/ ) {
16     $errors->{ 'phone' } = "Phone number incorrectly formatted";
17 }
18
19 my $values = [];
20 foreach ( @fields ) {
21     push( @$values, {
22         name => $_,
23         value => $params->{ $_ },
24         error => $errors->{ $_ },
25     } );
26 }
27
28 $tmpl->param( valueList => $values );
29
30 print $cgi->header();
31 print $tmpl->output;
32
33 46.6 Bot
  
```

- 8 Open the template.

- 9 Type `<tmpl_if name=error>` to check if there are errors as the `TMPL_LOOP` is processed.

- 10 Type `<tmpl_var name=error>` to display the error value to the user.

- 11 Type `</tmpl_if>`.

```

1 process-form.tmpl
2
3 <html>
4 <body>
5 <tmpl_if name=valueList>
6 <ul>
7 <tmpl_loop name=valueList>
8 <li><tmpl_var name=name>: <tmpl_var name=value>
9 <tmpl_if name=error>
10 ERROR: <tmpl_var name=error>
11 </tmpl_if>
12 </li>
13 </ul>
14 </tmpl_if>
15
16 <form action="process-form.pl" method=post>
17 <table>
18 <tr>
19 <td>First name:</td><td><input type=text name=firstname></td>
20 </tr>
21 <tr>
22 <td>Last name:</td><td><input type=text name=lastname></td>
23 </tr>
24 <tr>
25 <td>Phone #:</td><td><input type=text name=phone></td>
26 </tr>
27 <tr>
28 <td>Email:</td><td><input type=text name=email></td>
29 </tr>
30 </table>
31 <input type=submit name=process value="Submit Data">
32 </form>
33 </body>
34
35 12.15 Top
  
```

Apply It

In your code that validates phone numbers, you should not exclude international users. Although the two formats are not mutually compatible, you can still validate both by implementing two conditional tests. Only one format test actually needs to match to identify the number as valid.

TYPE THIS

```

if ( $phone =~ /\d{3}-\d{3}-\d{4}$/ ||
    $phone =~ /\^+\d+$/ ) {
    # This looks like a phone number!
}
  
```



RESULTS

Numbers formatted using either the North American or international syntax rules are accepted. However, North American numbers cannot use a plus sign and require dashes, while international numbers must begin with a plus sign and use no dashes.

Introducing the Apache mod_perl Module

You can enable the Apache mod_perl module as a way to embed the Perl interpreter directly within your Apache Web server. Doing this makes your CGI script's respond much faster because Apache has most of the Perl code preloaded into memory before the first user visits your Web site. If your CGI scripts import any Perl modules, mod_perl will keep them active in memory after your program is finished, so subsequent CGI load times will become faster.

Without mod_perl, each time a user types in your CGI script's URL, the Apache CGI handler reads the first line of the script to identify the file type and supporting

binary interpreter. This is why all your CGI scripts have either `#!/usr/bin/perl` or `#!C:/Perl/bin/perl.exe` at the top of every Perl file. The CGI handler executes the Perl binary, which then loads its internals into memory, interprets your script into binary code, and executes it. This process happens for every single CGI page request.

With mod_perl, Apache already has the Perl binary in memory, along with any support modules that your CGI script requires. The only thing it needs to do is interpret your script into binary code and execute it.

You can access online documentation for mod_perl 2.0 from the mod_perl homepage at <http://perl.apache.org/docs/2.0/index.html>.

Installing mod_perl

The mod_perl installation process is relatively simple. You need to install a new file called `mod_perl.so` into the Apache modules directory. Unfortunately, this file is not always delivered with the normal Apache installation method, depending on your platform.

On Windows, you can easily install `mod_perl.so` with the ActiveState Perl Package Manager (PPM). While it is not a part of the standard ActiveState installation, some community members have precompiled mod_perl's code

for Windows and have set up a PPM repository. This makes downloading and installing mod_perl very easy.

On Linux, `mod_perl.so` may be delivered as a standard part of your Apache installation, but this depends on your distribution. If you are using Debian or Ubuntu Linux, you need to install an additional package, `libapache2-mod-perl2`. If you are using a distribution based on Red Hat Linux, such as Fedora, mod_perl should already be installed alongside Apache; if not, you need to install the package `mod_perl`.

Activating mod_perl

The activation process is split into two steps: loading the module and applying it to `cgi-bin`.

Load mod_perl.so

You need to modify your Apache configuration to load `mod_perl.so` into memory. This process may be done automatically for you when you install mod_perl onto your Web server; however, if you notice that after completing this chapter you cannot use mod_perl, you should validate that the module has actually been loaded.

If you are using Windows, there are two commands to add into `httpd.conf`:

```
LoadFile C:/Perl/bin/perl510.dll
LoadModule perl_module modules/mod_perl.so
```

If you are using a Linux distribution, you only need one command in `httpd.conf`:

```
LoadModule perl_module /usr/lib/apache2/modules/
mod_perl.so
```

Note that the actual path to `mod_perl.so` on Linux should be absolute, and on Windows it is relative to the Apache installation directory.

Applying mod_perl to cgi-bin

In Chapter 10, you learned that in order to enable the Apache CGI handler onto a `cgi-bin` directory, you must add the following directive:

```
Options +ExecCGI
```

To activate `mod_perl`, you need to add three more new directives provided by `mod_perl.so`:

```
SetHandler perl-script
PerlResponseHandler ModPerl::Registry
PerlOptions +ParseHeaders
```

As described in Chapter 10, the `cgi-bin` directory can be defined in the original `httpd.conf` configuration file that is activated within a `<directory>` or `<location>` configuration group, or it can be applied in an `.htaccess` file in the `cgi-bin` directory. If you use the `httpd.conf` method, remember to restart the Apache service to re-read the new configuration.

Using mod_perl

Enabling `mod_perl` in your code is completely automatic; however, it does not behave like the normal Apache CGI handler. You may find that some bugs appear differently, or not at all, when comparing the two methods.

Speed Up Perl CGI Scripts

The Apache module that speeds up your Web site is called `ModPerl::Registry`. Its job is to read your Perl script, compile it, execute it, and store a cache in memory. Because Perl is constantly running in each Apache child process, every CGI page that was served by that process has a cache of your script in memory.

Technically speaking, there is nothing that you need to do to your Perl code to take advantage of `mod_perl`. Most Perl syntax is still valid; however, if you use “dirty” code — for example, by not declaring your variables with `my` — `mod_perl` may produce unexpected results when your CGI code assigns and retrieves these variables’ values.

The best way to reduce these anomalies is to import the `strict` module in all your CGI scripts and modules. This enforces proper variable declaration, as well as other good code habits:

```
use strict;
```

Preload Perl Modules

You can use `mod_perl` to preload modules into memory on Apache startup. All modules that are loaded will be kept persistently alive in memory, making them faster to access in your CGI code.

The only real disadvantage to `mod_perl` is that it can complicate things if you are actively developing any custom

Perl modules when it is enabled. If you change any of the module code, you need to restart the Apache service to force `mod_perl` to refresh itself. This is because the modules are cached by each Apache process serving HTTP requests. If a process has not seen your CGI script, it loads the latest module code; however, if it has served a request using an earlier version of your module, that earlier version will be maintained in memory and reused for future page hits.

To avoid this problem, you can implement the `Apache2::Reload` module in your Apache configuration:

```
PerlModule Apache2::Reload
PerlInitHandler Apache2::Reload
```

However, only do this on your development machine. Applying this onto your production network may result in a degraded performance gain each time the modules change. In this case, it is probably best just to restart the Apache server and avoid `Apache2::Reload`.

Other Features and Benefits

Other than Perl script and module speed-ups, `mod_perl` also allows you to write Apache handlers, and even create custom Apache directives out of raw Perl code.

A handler is developed like a Perl module, except that it is loaded directly by your Apache configuration. Once assigned to a directory path, all requests to that directory execute a special function in the handler.

Install the Apache mod_perl Module for Windows

You can install mod_perl for Windows by using a prebuilt binary package available from a third-party repository that is compatible with ActiveState Perl Package Manager (PPM). The installation process downloads the necessary Perl package descriptor (PPD) file from the on-line host and installs it with a single command.

The installation prompts you for the location of the Apache module directory. This is required so that the PPD knows where to save the `mod_perl.so` file that Apache will reference.

Once PPM is finished, you still need to instruct Apache to load the new `mod_perl.so` file, along with a Perl DLL. You do this by adding new `LoadFile` and `LoadModule` directives to the Apache `httpd.conf` file.

Extra

The third-party PPM repository located at the University of Winnipeg contains almost 300 other packages that you can use to extend ActiveState Perl. To enable the repository, you need to start PPM in Windows and click **Edit → Preferences → Repositories**.

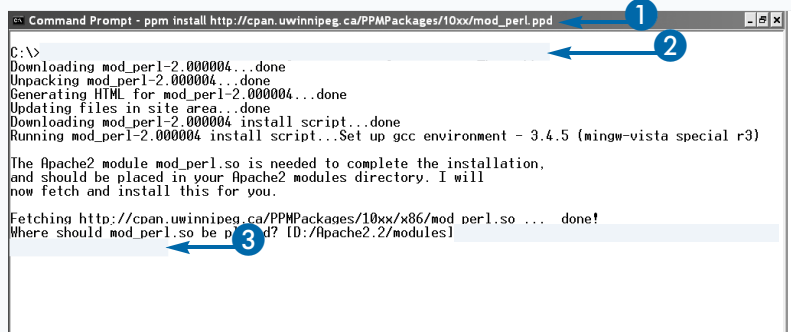
The Repositories screen comes preconfigured with suggested repositories, so you can just select `uwinnipeg :: University of Winnipeg` from the suggested drop-down list and click **Add**. After PPM resynchronizes its database, you can browse the packages provided by UWinnipeg PPM by clicking **View → View Columns → Repo**.

Install the Apache mod_perl Module for Windows

- 1 Open a DOS Prompt in Windows.
- 2 Type `ppm install http://cpan.uwinnipeg.ca/PPMPackages/10xx/mod_perl.ppd` and press Enter.
The `mod_perl` PPM downloads.
- 3 Type `C:\Program Files\Apache Software Foundation\Apache2.2\modules` and press Enter.

Note: You need to verify that this path is the actual directory where Apache 2.2 modules exist.

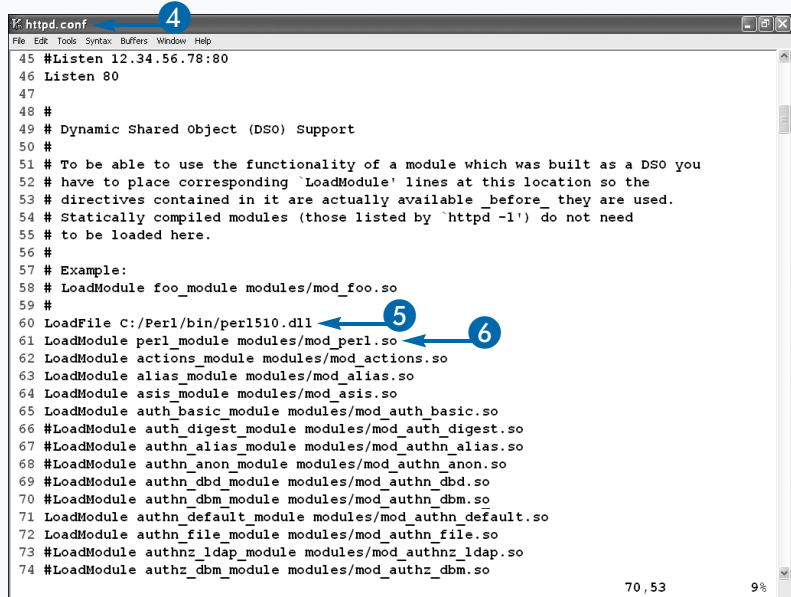
- 4 Open the Apache configuration file.
- 5 Type `LoadFile C:/Perl/bin/perl510.dll`.
- 6 Type `LoadModule perl_module modules/mod_perl.so`.
- 7 Save the configuration file and restart Apache.



```
Command Prompt - ppm install http://cpan.uwinnipeg.ca/PPMPackages/10xx/mod_perl.ppd
C:\>
Downloading mod_perl-2.000004...done
Unpacking mod_perl-2.000004...done
Generating HTML for mod_perl-2.000004...done
Updating files in site area...done
Downloading mod_perl-2.000004 install script...done
Running mod_perl-2.000004 install script...Set up gcc environment - 3.4.5 (mingw-vista special r3)

The Apache2 module mod_perl.so is needed to complete the installation,
and should be placed in your Apache2 modules directory. I will
now fetch and install this for you.

Fetching http://cpan.uwinnipeg.ca/PPMPackages/10xx/x86/mod_perl.so ... done!
Where should mod_perl.so be placed? [D:/Apache2.2/modules]
```



```
httpd.conf
File Edit Tools Syntax Buffers Window Help
45 #Listen 12.34.56.78:80
46 Listen 80
47
48 #
49 # Dynamic Shared Object (DSO) Support
50 #
51 # To be able to use the functionality of a module which was built as a DSO you
52 # have to place corresponding 'LoadModule' lines at this location so the
53 # directives contained in it are actually available before they are used.
54 # Statically compiled modules (those listed by 'httpd -l') do not need
55 # to be loaded here.
56 #
57 # Example:
58 # LoadModule foo_module modules/mod_foo.so
59 #
60 LoadFile C:/Perl/bin/perl510.dll
61 LoadModule perl_module modules/mod_perl.so
62 LoadModule actions_module modules/mod_actions.so
63 LoadModule alias_module modules/mod_alias.so
64 LoadModule asis_module modules/mod_asis.so
65 LoadModule auth_basic_module modules/mod_auth_basic.so
66 #LoadModule auth_digest_module modules/mod_auth_digest.so
67 #LoadModule authn_alias_module modules/mod_authn_alias.so
68 #LoadModule authn_anon_module modules/mod_authn_anon.so
69 #LoadModule authn_dbd_module modules/mod_authn_dbd.so
70 #LoadModule authn_dbm_module modules/mod_authn_dbm.so
71 LoadModule authn_default_module modules/mod_authn_default.so
72 LoadModule authn_file_module modules/mod_authn_file.so
73 #LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
74 #LoadModule authz_dbm_module modules/mod_authz_dbm.so
```

Install the Apache mod_perl Module for Linux

You can install mod_perl for Linux by using a prebuilt binary package that is available from your Linux distribution repository using either `apt-get` or `yum`. The installation process downloads the package from the repository and installs it with a single command. The installation process requires that your computer be connected to the Internet. The download, installation, and setup process should be executed in a Terminal window.

If you are using Debian or Ubuntu, the package you need to install is called `libapache2-mod-perl2`. Optionally, the mod_perl documentation is available in the package `libapache2-mod-perl2-doc`.

If you are using a Red Hat-based distribution, the

package you need to install is just called `mod_perl`. This package installation includes additional documentation by default.

Regardless of the Linux distribution, the mod_perl documentation is available under the directory `/usr/share/doc/packageName/`.

After you install the mod_perl package, the actual module file, `mod_perl.so`, should automatically load into your Apache configuration. All you need to do is restart Apache to enable it.

If `mod_perl.so` is not loading properly, you may need to manually add the `LoadModule` directive to specify the path to the module. Be sure to validate that `mod_perl.so` exists in this path:

```
LoadModule perl_module /usr/lib/apache2/
modules/mod_perl.so
```

Install the Apache mod_perl Module for Linux

- 1 Open a Terminal window in Linux.
- 2 (Debian/Ubuntu only) Type **`sudo apt-get install libapache2-mod-perl2`** and press Enter.

(Red Hat only) Type **`yum install mod_perl`** and press Enter.
- 3 Type **`Y`** and press Enter.

The mod_perl package downloads and installs.

- 4 (Debian/Ubuntu only) Type **`sudo /etc/init.d/apache2 restart`** to restart Apache.

(Red Hat only) Type **`su -c '/etc/init.d/httpd restart'`** and press Enter.

Apache restarts.

```
adamm@ubuntu: ~$ sudo apt-get install libapache2-mod-perl2
[sudo] password for adamm:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.31-14 libdns50 linux-headers-2.6.31-14-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  libapache2-reload-perl libbsd-resource-perl libdevel-syndump-perl
The following NEW packages will be installed:
  libapache2-mod-perl2 libapache2-reload-perl libbsd-resource-perl libdevel-syndump-perl
0 upgraded, 4 newly installed, 0 to remove and 7 not upgraded.
Need to get 1,186kB of archives.
After this operation, 4,272kB of additional disk space will be used.
Do you want to continue [Y/n]? 
```

```
adamm@ubuntu: ~$ sudo /etc/init.d/apache2 restart
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.31-14 libdns50 linux-headers-2.6.31-14-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  libapache2-reload-perl libbsd-resource-perl libdevel-syndump-perl
The following NEW packages will be installed:
  libapache2-mod-perl2 libapache2-reload-perl libbsd-resource-perl libdevel-syndump-perl
0 upgraded, 4 newly installed, 0 to remove and 7 not upgraded.
Need to get 0B/1,186kB of archives.
After this operation, 4,272kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Selecting previously deselected package libdevel-syndump-perl.
(Reading database ... 157069 files and directories currently installed.)
Unpacking libdevel-syndump-perl (from .../libdevel-syndump-perl_2.08-2_all.deb) ...
Selecting previously deselected package libapache2-mod-perl2.
Unpacking libapache2-mod-perl2 (from .../libapache2-mod-perl2_2.0.4-5ubuntu1_i386.deb) ...
Selecting previously deselected package libapache2-reload-perl.
Unpacking libapache2-reload-perl (from .../libapache2-reload-perl_0.10-2_all.deb) ...
Selecting previously deselected package libbsd-resource-perl.
Unpacking libbsd-resource-perl (from .../libbsd-resource-perl_1.2903-1_i386.deb) ...
Processing triggers for man-db ...
Setting up libdevel-syndump-perl (2.08-2) ...
Setting up libapache2-mod-perl2 (2.0.4-5ubuntu1) ...
Enabling module perl.
Run '/etc/init.d/apache2 restart' to activate new configuration!

Setting up libapache2-reload-perl (0.10-2) ...
Setting up libbsd-resource-perl (1.2903-1) ...
adamm@ubuntu: ~$
* Restarting web server apache2
... waiting ...
adamm@ubuntu: ~$
```

Configure the Apache mod_perl Module

You can configure mod_perl in Apache by adding a few new configuration directives into the Apache httpd.conf or .htaccess file. As discussed in Chapter 10, enabling the Apache CGI handler involves using Options +ExecCGI in either file. This is still required for mod_perl with a few new directives:

```
SetHandler perl-script
PerlResponseHandler ModPerl::Registry
PerlOptions +ParseHeaders
```

The SetHandler directive allows you to set all matching files to be processed by a specific handler. In essence, the value perl-script activates mod_perl, adding support for all new Apache directives that begin with “Perl”.

PerlResponseHandler defines which Perl module provides the actual HTTP response to the user. The value

ModPerl::Registry allows mod_perl to read the URL requested, identify which Perl CGI file is required, and forward your code to the Perl process.

The PerlOptions directive allows you to pass additional options specific to the response handler. In this example, +ParseHeaders is an instruction to ModPerl::Registry that the Perl script is called to produce additional HTTP headers that must be included in the Apache output to the user. You can validate whether or not mod_perl is working by reading some new environment variables within your Perl CGI scripts. Specifically, \$ENV{ 'MOD_PERL' } displays the version installed, and \$ENV{ 'MOD_PERL_API_VERSION' } reports the API version implemented. If either is absent, mod_perl has not been properly configured. A complete description of Apache directives provided by mod_perl is available at <http://perl.apache.org/docs/2.0/user/config/config.html>.

Configure the Apache mod_perl Module

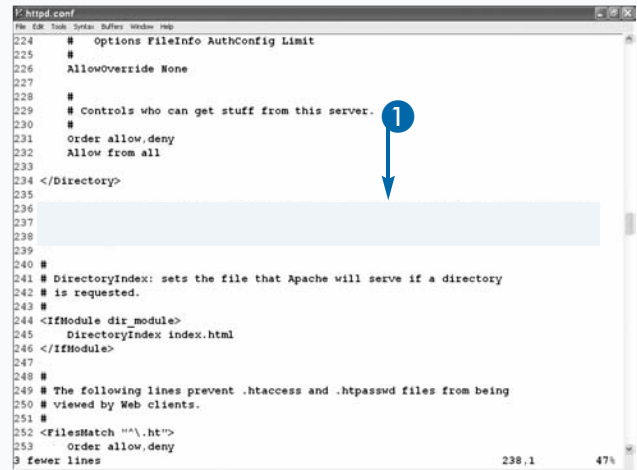
- 1 Open an Apache configuration file that already enables the CGI handler.

Note: See Chapter 10 for examples of enabling the CGI handler in <directory> and .htaccess contexts.

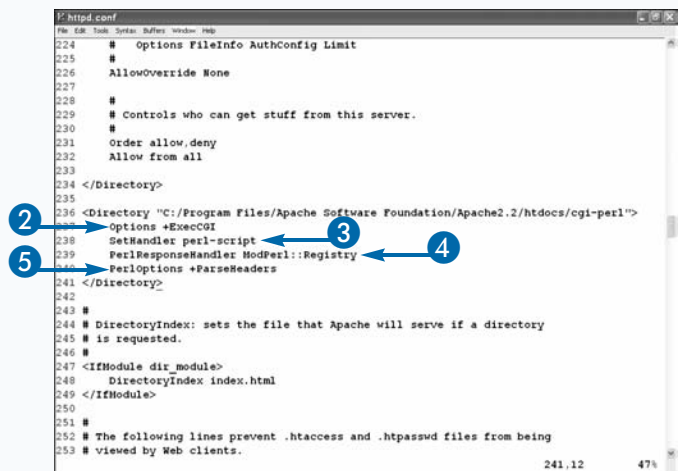
Note: Technically, using cgi-bin as a CGI directory name is no longer accurate. Here you call it cgi-perl, as only Perl scripts can be executed in a mod_perl directory.

- 2 Verify that Options +ExecCGI is already present, as described in Chapter 10.
- 3 Type SetHandler perl-script.
- 4 Type PerlResponseHandler ModPerl::Registry.
- 5 Type PerlOptions +ParseHeaders.
- 6 Save the configuration file.

Note: If you modified httpd.conf, you should restart Apache.



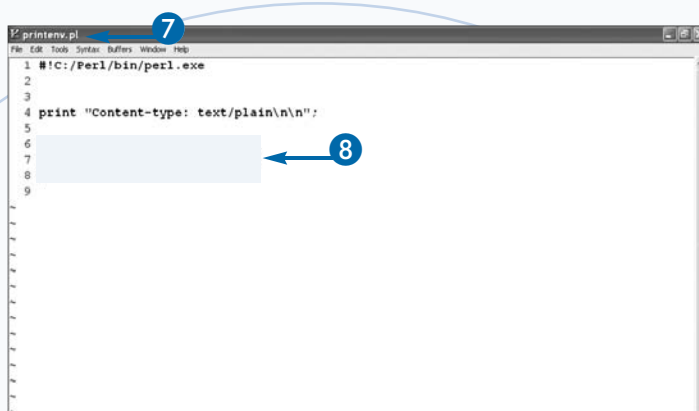
```
224 # Options FileInfo AuthConfig Limit
225 #
226 AllowOverride None
227
228 #
229 # Controls who can get stuff from this server.
230 #
231 Order allow,deny
232 Allow from all
233
234 </Directory>
235
236
237
238
239
240 #
241 # DirectoryIndex: sets the file that Apache will serve if a directory
242 # is requested.
243 #
244 <IfModule dir_module>
245     DirectoryIndex index.html
246 </IfModule>
247
248 #
249 # The following lines prevent .htaccess and .htpasswd files from being
250 # viewed by Web clients.
251 #
252 <FilesMatch "\.ht">
253     Order allow,deny
254     <Deny>
255     </Deny>
256 </FilesMatch>
```



```
224 # Options FileInfo AuthConfig Limit
225 #
226 AllowOverride None
227
228 #
229 # Controls who can get stuff from this server.
230 #
231 Order allow,deny
232 Allow from all
233
234 </Directory>
235
236 <Directory "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs/cgi-perl">
237     Options +ExecCGI
238     SetHandler perl-script
239     PerlResponseHandler ModPerl::Registry
240     PerlOptions +ParseHeaders
241 </Directory>
242
243 #
244 # DirectoryIndex: sets the file that Apache will serve if a directory
245 # is requested.
246 #
247 <IfModule dir_module>
248     DirectoryIndex index.html
249 </IfModule>
250
251 #
252 # The following lines prevent .htaccess and .htpasswd files from being
253 # viewed by Web clients.
```

- 7 Open a Perl CGI script in the `cgi-perl` directory.
- 8 Display environment variables from the HTTP session.

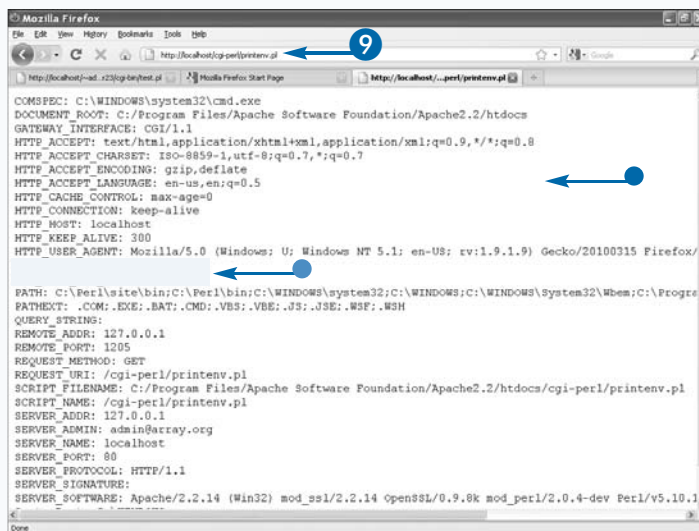
Note: Simply typing `print $ENV{'MOD_PERL'};` would be sufficient.



- 9 Open the Perl CGI script in a Web browser.

- The browser displays the current environment variables.
- The Apache `mod_perl` module is correctly enabled.

Note: If the `MOD_PERL` environment variable is not present, `mod_perl` has not been correctly enabled or installed.



Apply It

So far, you have learned that `mod_perl` delivers its content through the `PerlResponseHandler` directive. The handler module that is most often used, `ModPerl::Registry`, acts as a wrapper between your Perl script and the embedded Perl interpreter inside of Apache. However, there are handlers that you can enable, and you can even create your own custom response handler.

One response handler that ships with `mod_perl` is `Apache2::Status`. Enabling this handler dumps a complete report of `mod_perl`'s status and activity within Apache to your Web browser, allowing you to view the current state of `mod_perl`'s internals and memory.

TYPE THIS INTO THE APACHE CONFIGURATION FILE

```
<Location /perl-status>
    SetHandler perl-script
    PerlResponseHandler Apache2::Status
</Location>
```



RESULTS

After restarting the server, navigate to `http://localhost/perl-status`. A status screen summarizes the current state of `mod_perl` on Apache.

For more information about how `Apache2::Status` works, see its `PerlDoc` manual page. If you want to learn how to program your own custom response handler, see the `mod_perl` documentation at <http://perl.apache.org/docs/2.0/user/coding/coding.html>.

Understanding mod_perl's Caveats

You may notice that after enabling mod_perl, your Perl CGI code starts to behave differently, at least compared to the built-in Apache CGI handler. mod_perl keeps your scripts and modules active in memory after the CGI request is completed, as opposed to freeing the memory immediately after your script exits. While this does speed up response times significantly, you need to watch out for a new series of problems that require different fixes and workarounds.

The benefits of switching to mod_perl certainly outweigh any caveats, but knowing them in advance will make the development process easier. Unless you understand how mod_perl works, you may believe your computer is misbehaving.

Issues such as spontaneous Web site bugs, changes simply not appearing online, and errors not properly being logged are all issues that new developers to mod_perl quickly become frustrated with — and it's justified. Why waste the effort debugging unexplained, intermittent problems on your Web site, when you could be working on new content and features?

Beyond what is described here, if you still need help, the online mod_perl documentation is a great resource for additional troubleshooting techniques. Visit <http://perl.apache.org/docs/2.0/user/troubleshooting/troubleshooting.html>.

Disappearing, Reappearing Content

As you develop your Web site's dynamic CGI code, you may notice that most changes appear online right away, but sometimes your site spontaneously reverts to an earlier copy of your code, and back again.

Apache always has several child processes running parallel in memory, which helps when a spike of users visit, but only one process fulfills a single HTTP request at a time. When a user requests a CGI script, mod_perl loads the code, executes it, and caches a binary snapshot of the code in memory. If that same process receives another request for the same page, it reuses the cache rather than referring to the original CGI file. This results in a significant increase in the speed of your Web site.

However, if you use poor development practices when coding, such as not declaring variables correctly, then seemingly random errors may be hiding in the cached data, and subsequent requests may produce unexpected results. Each process has its own memory pool, so the cached data is not shared. When a different process responds to a request for the same CGI page, it will load the code directly from the file on your server, and everything will appear correct once again. The best way to avoid this problem is to write safe code, and import the `strict` module.

Restrict Unsafe Perl Code

You should import the `strict` module in your Perl code to reduce the risk of unsafe code problems, especially after enabling mod_perl. You can identify unsafe code by adding the following code to your scripts and modules:

```
use strict;
```

The `strict` module requires that all variables are properly declared with `my`. Not using `my` can cause earlier values to drift across HTTP requests. For example, if you have a Perl script that always sets a value to a variable, yet does not use `my`, it will work correctly. However, if your code only sets a value, then the variable is undefined, and unexpected things may happen to the variable and its value:

```
if ( ! defined $time ) {  
    $time = localtime;  
}  
print $time;
```

When Apache first runs the script, `$time` is set, but on subsequent reloads the `$time` value will never change. After importing the `strict` module, you will see an error such as, "Global symbol '`$time`' requires explicit page name", which is another way of saying, "Use '`my $time`' to localize your variable." The ModPerl::Registry handler, which understands `my`, respects the variable's localization and ignores its value when the script's cache is executed again.

Changes in Perl Files

Because `mod_perl` maintains a persistent Perl process in Apache, if any of your Perl scripts change, then `ModPerl::Registry` should load the new file into memory. However, Perl modules do not do this by default. Therefore, if your code changes frequently, you need to manage Apache and `mod_perl` accordingly.

Restarting Apache

You may consider simply restarting the entire Apache service to flush its memory of cached data. While this may seem excessive, it is not all bad, as Apache comes back quickly enough. Just be careful if your Web site offers large files to download, as the downloads could be interrupted when restarting.

Spawning New Apache Children

If you change a Perl module and do not restart Apache, the new module code will eventually be used on your Web site, but only after a new Apache child process has been spawned.

The Apache configuration determines how long each child process is to remain alive in memory before being restarted. The directive `MaxRequestsPerChild` is set rather high at 10000, and is mostly there as a catch-all to prevent accidental memory leakage within Apache.

For very minor module changes on a high-traffic Web site, this solution will have the least impact on the Web site's day-to-day responsiveness. Not restarting Apache means no downtime; however, if your traffic is not high enough, it may be a while before new module code naturally appears. For major module changes, you can either restart Apache or implement the `Apache2::Reload` module.

Monitoring Module Files, Reloading When Changed

You can instruct `mod_perl` to monitor modules used by Perl and to reload them in memory if they change on disk. There is a slight reduction in performance if you enable this feature, so you should only do so on development machines, or only enable it sparingly:

```
PerlModule Apache2::Reload
PerlInitHandler Apache2::Reload
```

This enables the reload functionality for all modules found in `@INC` and used in your Perl code. If you only want to limit `Apache2::Reload` to specific, frequently changing modules, you can specify them in your Apache configuration:

```
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "ModPerl::*
Apache2::*"

```

Alternatively, inside applicable Perl modules, you can import `Apache2::Reload` directly, which has the same effect. To validate that modules are correctly reloading when changed, you can enable additional debugging on `Apache2::Reload` with the following code:

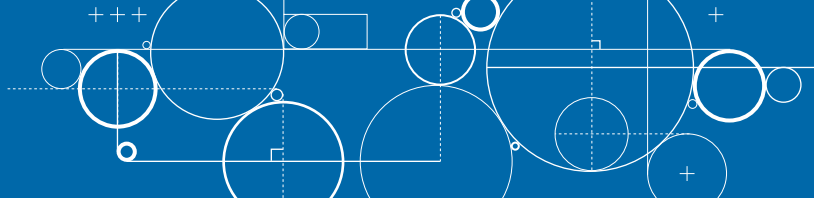
```
PerlSetVar ReloadDebug On
```

Disable `mod_perl` While Developing

You may want to consider disabling `mod_perl` while doing any major code development. Remember that `mod_perl` is best used as an efficiency tool in a production environment, and it can actually get in the way while you are actively developing, as described earlier.

Once your new feature stabilizes, and all bugs are removed, do not forget to test your code with `mod_perl` enabled before you deploy your update to production. As mentioned, there is always a chance that a problem will remain hidden under the regular CGI handler, but will become prevalent under `mod_perl`.

Access Perl Documentation



PerlDoc is a command-line program that you can use to look up Perl documentation in pod format and display it on your screen. All Perl modules, functions, operations, regular expressions, and command-line options are available in pod format, and are shipped with most standard Perl distributions. A complete list of

PerlDoc pages is available on the main Perl documentation page. You can access this page by running the following commands on a Terminal window or DOS Prompt:

```
perldoc perl.
```

You can also access the complete PerlDoc library online at www.perl.com/pub/q/documentation.

COMMON COMMANDS	DESCRIPTION
<code>perldoc <i>page</i></code>	View the documentation of a page.
<code>perldoc <i>module</i></code>	View the documentation of a module.
<code>perldoc <i>program</i></code>	View the documentation of a program.
<code>perldoc -f <i>function</i></code>	View the documentation of a function.
<code>perldoc -q <i>keyword</i></code>	Search the FAQ by <i>keyword</i> .

PERLDOC OPTIONS	DESCRIPTION
<code>-h</code>	View the help page.
<code>-v</code>	Display verbose details when searching.
<code>-t</code>	Display documentation pages in plain-text.
<code>-u</code>	Show the unformatted <code>.pod</code> source.
<code>-l</code>	Display only the filename found.
<code>-F <i>file</i></code>	Open a specific <code>file.pod</code> file.
<code>-L <i>code</i></code>	Open <code>.pod</code> files by language-code.
<code>-V</code>	Display the version of PerlDoc.

OVERVIEW PAGE	DESCRIPTION
<code>perl</code>	Display an overview of Perl.
<code>perlintro</code>	Display an introduction to Perl for beginners.
<code>perltoc</code>	Display the Perl documentation table of contents.

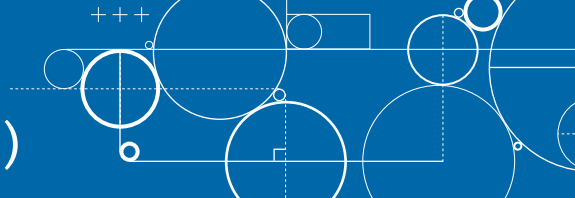
TUTORIAL PAGE	DESCRIPTION
<code>perlreftut</code>	Perl references short introduction
<code>perldsc</code>	Perl data structures intro
<code>perllo1</code>	Perl data structures: arrays of arrays

TUTORIAL PAGE	DESCRIPTION
perlrequick	Perl regular expressions introduction
perlretut	Perl regular expressions tutorial
perlboot	Perl object-orientated tutorial for beginners
perltoot	Perl object-orientated tutorial, part 1
perltooc	Perl object-orientated tutorial, part 2
perlbot	Perl object-orientated tricks and examples
perlstyle	Perl style guide
perlcheat	Perl cheat sheet
perltrap	Perl traps for the unwary
perldebtut	Perl debugging tutorial
perlopentut	Perl open tutorial
perlpacktut	Perl pack and unpack tutorial
perlthrtut	Perl threads tutorial
perlfaq	Perl FAQ table of contents
perlfaq1	FAQ: General Questions about Perl
perlfaq2	FAQ: Obtaining and Learning about Perl
perlfaq3	FAQ: Programming Tools
perlfaq4	FAQ: Data Manipulation
perlfaq5	FAQ: Files and Formats
perlfaq6	FAQ: Regular Expressions
perlfaq7	FAQ: Perl Language Issues
perlfaq8	FAQ: System Interaction
perlfaq9	FAQ: Networking

REFERENCE PAGE	DESCRIPTION
perlsyn	Perl syntax
perldata	Perl data structures
perlop	Perl operators and precedence
perlsub	Perl subroutines
perlfunc	Perl built-in functions
perlpod	Perl plain old documentation
perlrun	Perl execution and options
perldiag	Perl diagnostic messages

(continued)

Access Perl Documentation (continued)



REFERENCE PAGE	DESCRIPTION
perllexwarn	Perl lexical warnings
perldebug	Perl debugging
perlvarz	Perl predefined variables
perlre	Perl regular expressions
perlrebackslash	Perl regular expressions: backslash sequences
perlrecharclass	Perl regular expressions: character classes
perlrefref	Perl regular expressions: quick reference
perlref	Perl references
perldata	Perl formats
perllobj	Perl objects
perltime	Perl objects hidden behind simple variables
perlipc	Perl interprocess communication
perlfork	Perl <code>fork</code> information
perlnumber	Perl number semantics
perlport	Perl portability guide
perllocale	Perl locale support
perluniintro	Perl Unicode introduction
perlunicode	Perl Unicode support
perlunifaq	Perl Unicode FAQ
perlunitut	Perl Unicode tutorial
perlsec	Perl security
perlmod	Perl modules: how they work
perlmodlib	Perl modules: how to write and use
perlmodstyle	Perl modules: how to write modules with style
perlmodinstall	Perl modules: how to install from CPAN
perlnewmod	Perl modules: preparing a new module for distribution
perlpragma	Perl modules: writing a user pragma
perlutil	Utilities packaged with the Perl distribution
perlcompile	Perl compiler suite introduction
perlfilter	Perl source filters
perlglossary	Perl glossary

Execute Perl on the Command-Line

The Perl interpreter allows for several command-line options, which enables you to change how Perl executes your programs. You can even execute Perl code as a *one-liner program* completely from the command-line. A one-liner is a simple Perl

program that is described and executed totally on the command-line, without the use of a .pl Perl script. You can find a complete list of available options on the following PerlDoc page using the command-line `perldoc perlrun`.

OPTION	DESCRIPTION
-c	Checks program syntax and exit.
-d	Runs the program under the Perl debugger.
-e <i>command</i>	Enters a Perl statement to be executed.
-h	Prints a summary of all options.
-i[<i>extension</i>]	Allows you to edit files in-place, optionally backing them to <i>extension</i> . This is useful with -p and -e.
-I <i>directory</i>	Adds <i>directory</i> into the search path for modules.
-M <i>module</i>	Imports a module into Perl. This is useful with -e.
-n	Loops around your one-line program by creating a <code>while (<>)</code> block for each line of input, similar to <code>sed -n</code> , or <code>awk</code> . This is useful with -e.
-p	Loops around your one-line program, just like <code>sed</code> , and prints the output when finished. This is useful with -e.
-T	Enables taint checks as fatal errors. Your program will not run if a security check is identified.
-t	Enables taint checks as warnings.
-v	Prints the version and patch-level of your Perl executable.
-V	Prints a summary of the major Perl configuration values.
-w	Prints warnings about dubious constructs and unusually formatted code.

Available Built-In Perl Functions

Perl functions refer to the built-in programs that you can call from within your Perl scripts and modules. More than 250 functions are available that allow you to manipulate scalars, arrays, hashes, numbers, lists, regular expressions, files, directories, sockets, system processes, and time. Even if a feature you are looking for is not available, there is probably a third-party module on CPAN that provides it.

For all the available Perl functions, the use of parenthesis — also referred to as round brackets — around the arguments are generally recommended for clarity, but are not required. Some functions enforce no parenthesis, such as `use`, but most functions and subroutines are flexible. For example, take the following program that uses all parentheses:

```
+++
open( H, ">> myfile.txt" ) || die( "error: $!"
);
print( H "The time is: " . time . "\n" );
close( H );
```

Now, it is rewritten without any parenthesis around its functions:

```
open H, ">> myfile.txt" || die "error: $!";
print H "The time is: " . time . "\n";
close H;
```

Both programs are technically identical, but appear very different. Deciding whether to use parenthesis around arguments is a matter of personal preference. Following are descriptions of several conventions that you can use to represent function arguments. See Chapter 6 for more information on these variable types.

CONVENTION	DESCRIPTION
<i>SCALAR</i>	A single string, number, object, or reference variable.
<i>ARRAY</i>	An array variable.
<i>HASH</i>	A hash variable.
<i>STRING</i>	A quoted string, or scalar holding a string.
<i>NUMBER</i>	A number.
<i>POSITION, OFFSET, LENGTH</i>	A number, usually counting from the start of a string or file handle referenced according to the function.
<i>EXPR</i>	An expression representing an object, such as a variable, quoted string, number, subroutine, and so on.
<i>LIST</i>	A series of zero or more expressions, comma-separated, or an array or series of arrays.
<i>TEMPLATE, FORMAT</i>	Predefined formatting specific to a function.
<i>DIRHANDLE, FILEHANDLE</i>	A handle used to read from a directory or file.
<i>PATTERN</i>	A regular-expression pattern.
<i>SUBNAME</i>	The name of a user-defined subroutine.
<i>BLOCK</i>	A { ... } block of code.

You can access additional details and documentation with the `Perldoc` program on the command-line using the following command:

```
perldoc -f function
```


SCALAR FUNCTIONS	DESCRIPTION
<code>chop(SCALAR)</code>	Chops off the last character of a string and returns the chopped character.
<code>chomp(SCALAR)</code>	A safer version of <code>chop</code> that removes any trailing input record separators, also known as line-feeds.
<code>chr(NUMBER)</code>	Returns the character represented by that <i>NUMBER</i> in the character set.
<code>crypt(STRING, EXPR)</code>	A one-way hash function that creates a digest string exactly like the <code>crypt(3)</code> function in the C library.
<code>index(STRING, SUBSTRING)</code>	Searches for one string within another.
<code>index(STRING, SUBSTRING, POSITION)</code>	Searches for one string within another, starting at <i>POSITION</i> .
<code>lc(EXPR)</code>	Returns a lowercased version of <i>EXPR</i> .
<code>lcfirst(EXPR)</code>	Returns the value of <i>EXPR</i> with the first character lowercased.
<code>length(EXPR)</code>	Returns the length in characters of the value of <i>EXPR</i> .
<code>oct(EXPR)</code>	Interprets <i>EXPR</i> as an octal string and returns the corresponding value.
<code>ord(EXPR)</code>	Returns the numeric value of the first character of <i>EXPR</i> .
<code>pack(TEMPLATE, LIST)</code>	Takes a <i>LIST</i> of values and converts it into a string using the rules given by the <i>TEMPLATE</i> .
<code>reverse(LIST)</code>	Concatenates the elements of <i>LIST</i> and returns a string value with all characters in the opposite order, when used to populate a <i>SCALAR</i> variable. May also be used in list context to populate an <i>ARRAY</i> variable.
<code>rindex(STRING, SUBSTRING)</code>	Works just like <code>index</code> except that it returns the position of the last occurrence of <i>SUBSTR</i> in <i>STR</i> .
<code>rindex(STRING, SUBSTRING, POSITION)</code>	Works like <code>rindex</code> but starts at <i>POSITION</i> .
<code>sprintf(FORMAT, LIST)</code>	Returns a string formatted by the usual <code>printf</code> conventions of the C library function <code>sprintf</code> .
<code>substr(EXPR, OFFSET)</code>	Extracts a substring out of <i>EXPR</i> and returns it.
<code>substr(EXPR, OFFSET, LENGTH)</code>	Extracts a substring out of <i>EXPR</i> , up to <i>LENGTH</i> , and returns it.
<code>substr(EXPR, OFFSET, LENGTH, REPLACEMENT)</code>	Extracts a substring out of <i>EXPR</i> , up to <i>LENGTH</i> , and replaces it with <i>REPLACEMENT</i> , returning the original match.
<code>uc(EXPR)</code>	Returns an uppercased version of <i>EXPR</i> .
<code>ucfirst(EXPR)</code>	Returns the value of <i>EXPR</i> with the first character in uppercase.

REGULAR EXPRESSION FUNCTIONS	DESCRIPTION
<code>pos(SCALAR)</code>	Returns the offset of where the last "m/ /g" search left off for the variable in question.
<code>quotemeta(EXPR)</code>	Returns the value of <i>EXPR</i> with all non-word characters backslashed.
<code>split(/PATTERN/, EXPR)</code>	Splits the string <i>EXPR</i> into a list of strings and returns that list.
<code>split(/PATTERN/, EXPR, LIMIT)</code>	Splits the string <i>EXPR</i> into a list of strings and returns that list, stopping at <i>LIMIT</i> .
<code>study(SCALAR)</code>	Takes extra time to study <i>SCALAR</i> in anticipation of doing many pattern matches on the string before it is next modified.



Available Built-In Perl Functions (continued)

NUMERIC FUNCTIONS	DESCRIPTION
<code>abs(NUMBER)</code>	Returns the absolute value of its argument.
<code>atan2(NUMBER1, NUMBER2)</code>	Returns the arctangent of numbers 1 and 2, in the range -PI to PI.
<code>cos(EXPR)</code>	Returns the cosine of <i>EXPR</i> (expressed in radians).
<code>exp(EXPR)</code>	Returns e (the natural logarithm base) to the power of <i>EXPR</i> .
<code>hex(EXPR)</code>	Interprets <i>EXPR</i> as a hexadecimal string and returns the corresponding decimal value.
<code>int(EXPR)</code>	Returns the integer portion of <i>EXPR</i> .
<code>log(EXPR)</code>	Returns the natural logarithm (base e) of <i>EXPR</i> .
<code>oct(EXPR)</code>	Interprets <i>EXPR</i> as an octal string and returns the corresponding value.
<code>rand(EXPR)</code>	Returns a random fractional number greater than or equal to zero and less than the value of <i>EXPR</i> .
<code>sin(EXPR)</code>	Returns the sine of <i>EXPR</i> (expressed in radians).
<code>sqrt(EXPR)</code>	Returns the square root of <i>EXPR</i> .
<code>srand(EXPR)</code>	Sets the random number seed for the <code>rand</code> operator.

ARRAY FUNCTIONS	DESCRIPTION
<code>pop(ARRAY)</code>	Returns the last value of the array, shortening the array by one element.
<code>push(ARRAY, LIST)</code>	Treats <i>ARRAY</i> as a stack, and inserts the values of <i>LIST</i> onto the end of <i>ARRAY</i> .
<code>shift(ARRAY)</code>	Returns the first value of the array, shortening the array by 1 and moving everything down.
<code>splice(ARRAY, OFFSET)</code>	Removes the elements designated by <i>OFFSET</i> from <i>ARRAY</i> .
<code>splice(ARRAY, OFFSET, LENGTH)</code>	Removes <i>LENGTH</i> number of elements designated by <i>OFFSET</i> from <i>ARRAY</i> .
<code>splice(ARRAY, OFFSET, LENGTH, LIST)</code>	Removes <i>LENGTH</i> number of elements designated by <i>OFFSET</i> from <i>ARRAY</i> , and replaces them with the elements of <i>LIST</i> .
<code>unshift(ARRAY, LIST)</code>	Treats <i>ARRAY</i> as a stack, and inserts the values of <i>LIST</i> onto the beginning of <i>ARRAY</i> .

LIST FUNCTIONS	DESCRIPTION
<code>grep(<i>EXPR</i>, <i>LIST</i>)</code>	Evaluates <i>EXPR</i> for each element of <i>LIST</i> (locally setting <code>\$_</code> to each element) and returns the list value consisting of those elements for which the expression evaluated to true.
<code>join(<i>EXPR</i>, <i>LIST</i>)</code>	Joins the separate strings of <i>LIST</i> into a single string with fields separated by the value of <i>EXPR</i> , and returns that new string.
<code>map(<i>EXPR</i>, <i>LIST</i>)</code>	Evaluates <i>EXPR</i> for each element of <i>LIST</i> (locally setting <code>\$_</code> to each element) and returns the list value composed of the results of each such evaluation.
<code>reverse(<i>LIST</i>)</code>	Returns a list value consisting of the elements of <i>LIST</i> in the opposite order, when used to populate an <i>ARRAY</i> variable. This function may also be used in scalar context to populate a <i>SCALAR</i> variable.
<code>sort(<i>SUBNAME LIST</i>)</code>	Sorts <i>LIST</i> using the subroutine <i>SUBNAME</i> for comparison logic and returns the sorted list value.
<code>sort(<i>BLOCK LIST</i>)</code>	Sorts <i>LIST</i> using an anonymous subroutine, or <i>BLOCK</i> , for comparison logic and returns the sorted list value.
<code>sort(<i>LIST</i>)</code>	Sorts <i>LIST</i> in standard string comparison order and returns the sorted list value.
<code>unpack(<i>TEMPLATE</i>, <i>EXPR</i>)</code>	Does the reverse of <code>pack</code> , taking a string and expanding it out into a list of values.

HASH FUNCTIONS	DESCRIPTION
<code>delete(<i>EXPR</i>)</code>	Given an expression that specifies a hash element, array element, hash slice, or array slice, deletes the specified element(s) from the hash or array.
<code>each(<i>HASH</i>)</code>	Returns a two-element list consisting of the key and value for the next element of a hash, so that you can iterate over it.
<code>exists(<i>EXPR</i>)</code>	Given an expression that specifies a hash element or array element, returns true if the specified element in the hash or array has ever been initialized, even if the corresponding value is undefined.
<code>keys(<i>HASH</i>)</code>	Returns a list consisting of all the keys of <i>HASH</i> .
<code>values(<i>HASH</i>)</code>	Returns a list consisting of all the values of <i>HASH</i> .

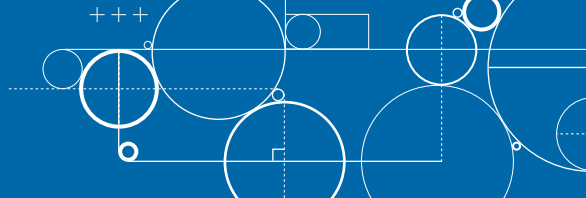
INPUT/OUTPUT FUNCTIONS	DESCRIPTION
<code>binmode(<i>FILEHANDLE</i>)</code>	Arranges for <i>FILEHANDLE</i> to be read or written in binary or text mode on systems where the run-time libraries distinguish between binary and text files.
<code>close(<i>FILEHANDLE</i>)</code>	Closes the file or pipe associated with the filehandle, flushes the IO buffers, and closes the system file descriptor.
<code>closedir(<i>DIRHANDLE</i>)</code>	Closes a directory opened by <code>opendir</code> .
<code>die(<i>LIST</i>)</code>	Prints the value of <i>LIST</i> to <code>STDERR</code> and exits with the current value of <code>\$!</code> .
<code>eof(<i>FILEHANDLE</i>)</code>	Returns 1 if the next read on <i>FILEHANDLE</i> will return end of file, or if <i>FILEHANDLE</i> is not open.
<code>fileno(<i>FILEHANDLE</i>)</code>	Returns the file descriptor for a filehandle, or undefined if the filehandle is not open.

(continued)

continued



Available Built-In Perl Functions (continued)



INPUT/OUTPUT FUNCTIONS	DESCRIPTION
<code>flock(FILEHANDLE, OPERATION)</code>	Calls the GNU C library's <code>flock</code> function, or an emulation of it, on <i>FILEHANDLE</i> . This is the Perl portable file locking interface, although it locks only entire files, not records.
<code>getc(FILEHANDLE)</code>	Returns the next character from the input file attached to <i>FILEHANDLE</i> .
<code>print(FILEHANDLE LIST)</code>	Prints a string or a list of strings to <i>FILEHANDLE</i> .
<code>print(LIST)</code>	Prints a string or a list of strings to <i>STDOUT</i> .
<code>printf(FILEHANDLE FORMAT, LIST)</code>	Uses <i>FORMAT</i> to print <i>LIST</i> to <i>FILEHANDLE</i> .
<code>printf(FORMAT, LIST)</code>	Uses <i>FORMAT</i> to print <i>LIST</i> to <i>STDOUT</i> .
<code>read(FILEHANDLE, SCALAR, LENGTH)</code>	Attempts to read <i>LENGTH</i> characters of data into variable <i>SCALAR</i> from the specified <i>FILEHANDLE</i> .
<code>read(FILEHANDLE, SCALAR, LENGTH, OFFSET)</code>	Like <code>read</code> but starts reading at <i>OFFSET</i> .
<code>readdir(DIRHANDLE)</code>	Returns the next directory entry for a directory opened by <code>opendir</code> .
<code>rewinddir(DIRHANDLE)</code>	Sets the current position to the beginning of the directory for the <code>readdir</code> routine on <i>DIRHANDLE</i> .
<code>say(LIST)</code>	Just like <code>print</code> , but automatically appends a new-line character.
<code>seek(FILEHANDLE, POSITION, WHENCE)</code>	Sets the current position for the <code>read</code> on <i>FILEHANDLE</i> , using <i>WHENCE</i> to define how the <i>POSITION</i> value changes the current position. The values for <i>WHENCE</i> are 0 to set the new position in bytes to <i>POSITION</i> , 1 to set it to the current position plus <i>POSITION</i> , and 2 to set it to <i>FILEHANDLE</i> 's end-of-file, plus <i>POSITION</i> .
<code>seekdir(DIRHANDLE, POSITION)</code>	Sets the current position for the <code>readdir</code> routine on <i>DIRHANDLE</i> .
<code>select(FILEHANDLE)</code>	Sets <i>FILEHANDLE</i> as the default filehandle for output.
<code>select</code>	Returns the currently selected filehandle.
<code>syscall(NUMBER, LIST)</code>	Calls the system call specified as the first element of the list, passing the remaining elements as arguments to the system call.
<code>sysread(FILEHANDLE, SCALAR, LENGTH)</code>	Attempts to read <i>LENGTH</i> bytes of data into variable <i>SCALAR</i> from the specified <i>FILEHANDLE</i> , using the GNU C library's <code>read</code> function.
<code>sysread(FILEHANDLE, SCALAR, LENGTH, OFFSET)</code>	Starts <code>sysread</code> at <i>OFFSET</i> bytes from <i>FILEHANDLE</i> .
<code>sysseek(FILEHANDLE, POSITION, WHENCE)</code>	Like <code>seek</code> , but uses the GNU C library's <code>lseek</code> function.
<code>tell(FILEHANDLE)</code>	Returns the current position in bytes for <i>FILEHANDLE</i> .
<code>telldir(DIRHANDLE)</code>	Returns the current position of the <code>readdir</code> routines on <i>DIRHANDLE</i> .
<code>truncate(FILEHANDLE, LENGTH)</code>	Truncates the file opened on <i>FILEHANDLE</i> to the specified length.
<code>warn(LIST)</code>	Prints the value of <i>LIST</i> to <i>STDERR</i> .

FILE/DIRECTORY FUNCTIONS	DESCRIPTION
<code>-X FILEHANDLE</code>	A file test, where X is a specific letter. See <code>perlfunc -f -X</code> for a valid letter.
<code>chdir(EXPR)</code>	Changes the working directory to <i>EXPR</i> .
<code>chmod(MODE, LIST)</code>	Changes the permissions of a list of files.
<code>chown(UID, GID, LIST)</code>	Changes the owner (and group) of a list of files.
<code>chroot(DIRNAME)</code>	Makes the named directory the new root directory for all further pathnames that begin with a forward slash (/) by your process and all its children.
<code>fcntl(FILEHANDLE, FUNCTION, SCALAR)</code>	Implements the GNU C library's <code>fcntl</code> function from Unix.
<code>glob(EXPR)</code>	In list context, returns a list of filename expansions on the value of <i>EXPR</i> such as the standard Unix shell would do.
<code>ioctl(FILEHANDLE, FUNCTION, SCALAR)</code>	Implements the GNU C library's <code>ioctl</code> function from Unix.
<code>link(OLDFILE, NEWFILE)</code>	Creates a new filename linked to the old filename.
<code>lstat(FILENAME)</code>	Does the same thing as the <code>stat</code> function but stats a symbolic link instead of the file that the symbolic link points to.
<code>mkdir(FILENAME, MASK)</code>	Creates the directory specified by <i>FILENAME</i> , with permissions specified by <i>MASK</i> (as modified by <code>umask</code>).
<code>open(FILEHANDLE, FILENAME)</code>	Opens the file whose filename is given by <i>EXPR</i> , and associates it with <i>FILEHANDLE</i> .
<code>opendir(DIRHANDLE, DIRNAME)</code>	Opens a directory named <i>EXPR</i> and associates it with <i>DIRHANDLE</i> .
<code>readlink(FILENAME)</code>	Returns the value of a symbolic link, if symbolic links are implemented.
<code>rename(OLDNAME, NEWNAME)</code>	Changes the name of a file; if an existing file is already using <i>NEWNAME</i> , it will be deleted.
<code>rmdir(DIRNAME)</code>	Deletes the directory specified by <i>DIRNAME</i> if that directory is empty.
<code>stat(HANDLE)</code>	Returns a 13-element list giving the status information for a file opened using <i>FILEHANDLE</i> or <i>DIRHANDLE</i> .
<code>stat(FILENAME)</code>	Returns a 13-element list giving the file-system status information for <i>FILENAME</i> .
<code>symlink(OLDFILE, NEWFILE)</code>	Creates a new filename symbolically linked to the old filename.
<code>sysopen(FILEHANDLE, FILENAME, MODE)</code>	Opens the file whose filename is given by <i>FILENAME</i> , and associates it with <i>FILEHANDLE</i> .
<code>umask(EXPR)</code>	Sets the <code>umask</code> for the process to <i>EXPR</i> and returns the previous value.
<code>unlink(LIST)</code>	Deletes a list of files.
<code>utime(LIST)</code>	Changes the access and modification times on each file of a list of files.

Available Built-In Perl Functions (continued)

+++

FLOW CONTROL KEYWORDS	DESCRIPTION
caller(<i>EXPR</i>)	Returns the context of the current subroutine call.
continue	Used to increment a loop variable within a block.
do(<i>FILENAME</i>)	Executes a Perl <i>FILENAME</i> within an existing Perl script.
dump	Causes an immediate core dump.
eval(<i>EXPR</i>)	The return value of <i>EXPR</i> is parsed and executed as if it were a short Perl program.
exit(<i>EXPR</i>)	Exits immediately with the value <i>EXPR</i> .
goto(<i>LABEL</i>)	The goto- <i>LABEL</i> form finds the statement labeled with <i>LABEL</i> and resumes execution there.
last	Like the break statement in C (as used in loops); it immediately exits the current loop.
next	Like the continue statement in C; it starts the next iteration of the loop.
redo	Restarts the loop block without evaluating the conditional expression again.
return(<i>EXPR</i>)	Returns from a subroutine, eval, or do with the value given in <i>EXPR</i> .
sub <i>NAME BLOCK</i>	Forms a subroutine definition.
wantarray	Returns true if the context of the currently executing subroutine or eval is looking for a list value.

MISCELLANEOUS FUNCTIONS	DESCRIPTION
defined(<i>EXPR</i>)	Returns a Boolean value telling whether <i>EXPR</i> has a value other than the undefined value undef.
local(<i>LIST</i>)	Modifies the listed variables to be local to the enclosing block, file, or eval.
my(<i>LIST</i>)	Declares the listed variables to be local (lexically) to the enclosing block, file, or eval.
our(<i>EXPR</i>)	Associates a simple name with a package variable in the current package for use within the current scope.
reset(<i>EXPR</i>)	Generally used in a continue block at the end of a loop to clear variables and reset ?? searches so that they work again.
scalar(<i>EXPR</i>)	Forces <i>EXPR</i> to be interpreted in scalar context and returns the value of <i>EXPR</i> .
undef(<i>EXPR</i>)	Undefines the value of <i>EXPR</i> .

PROCESS FUNCTIONS	DESCRIPTION
<code>alarm(SECONDS)</code>	Arranges to have a <i>SIGALRM</i> delivered to this process after the specified number of seconds has elapsed.
<code>exec(PROGRAM, LIST)</code>	Executes a system command <i>PROGRAM</i> with arguments <i>LIST</i> and never returns.
<code>fork</code>	Executes the GNU C library's <i>fork</i> function to create a new process running the same program at the same point.
<code>kill(SIGNAL, LIST)</code>	Sends a signal to a list of processes.
<code>pipe(READHANDLE, WRITEHANDLE)</code>	Opens a pair of connected pipes like the corresponding system call.
<code>sleep(EXPR)</code>	Causes the script to sleep for <i>EXPR</i> seconds, or forever if there is no <i>EXPR</i> .
<code>system(PROGRAM, LIST)</code>	Executes a system command <i>PROGRAM</i> with arguments <i>LIST</i> .
<code>wait</code>	Waits for a child process to terminate and returns the <i>pid</i> of the deceased process.
<code>waitpid(PID, FLAGS)</code>	Waits for a particular child process to terminate and returns the <i>pid</i> of the deceased process.

MODULE KEYWORDS	DESCRIPTION
<code>bless(REF, CLASSNAME)</code>	Tells <i>REF</i> that it is now an object in the <i>CLASSNAME</i> package.
<code>no MODULE</code>	The opposite of <i>use</i> . This is used to unload a module.
<code>package NAME</code>	Declares the compilation unit as being in the given namespace.
<code>require VERSION</code>	Demands a minimum version of Perl specified by <i>VERSION</i> .
<code>use MODULE LIST</code>	Imports some semantics into the current package from the named module, generally by aliasing certain subroutine or variable names into your package.

TIME FUNCTIONS	DESCRIPTION
<code>gmtime(EXPR)</code>	Works just like <i>localtime</i> but the returned values are localized for the standard Greenwich time zone.
<code>localtime(EXPR)</code>	Converts a time as returned by the time function to a nine-element list with the time analyzed for the local time zone. In scalar context, it returns the GNU C library's <i>ctime</i> value string.
<code>time</code>	Returns the number of non-leap seconds since whatever time the system considers to be the epoch.
<code>times</code>	Returns a four-element list giving the user and system times, in seconds, for this process and the children of this process.

Using Perl Pre-Defined Variables

Perl supports a series of pre-defined variables that you can use within your scripts. Most of these variables are read-only, but you can use them in various contexts of your Perl code to customize your Perl experience. Many pre-defined variables are only useful under specific contexts. Some are rarely used, while others are more common. For example, `$_` is commonly used as the default iterator within `foreach` and `while` loops, where each element within can be accessed with this variable.

As you go through this table, do not think of it as a list of everything you should be taking advantage of, but

instead keep it in mind as you read existing Perl code. If a program arbitrarily assigns or retrieves a variable that does not appear to be properly declared, you should be able to find that variable, and an explanation of what it does, in this table. Once you are familiar with what is available, you may come across a situation where changing one of these pre-defined variables is legitimately warranted.

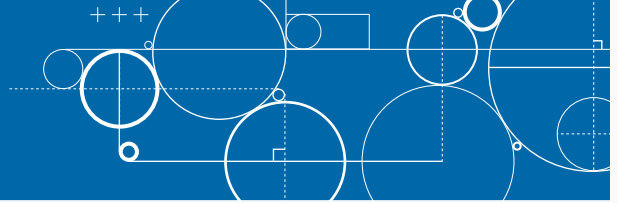
You can access additional details and documentation with the `Perldoc` program on the command-line using `perldoc perlvar`.

VARIABLE	DESCRIPTION
<code>\$_</code>	The default input and pattern-searching space.
<code>\$a, \$b</code>	Special package variables when using <code>sort</code> .
<code>\$digit</code>	Contains the subpattern from the corresponding set of capturing parentheses from the last regular expression pattern match.
<code>\$&, \$MATCH</code>	The string matched by the last successful pattern match.
<code>\$`, \$PREMATCH</code>	The string preceding whatever was matched by the last successful pattern match.
<code>\$', \$POSTMATCH</code>	The string following whatever was matched by the last successful pattern match.
<code>\$/, \$LAST_PAREN_MATCH</code>	The text matched by the last parenthesis of the last successful regular expression search pattern.
<code>\$\$N, \$LAST_SUBMATCH_RESULT</code>	The text matched by the used group that was most-recently closed in the last successful search pattern.
<code>@+, @LAST_MATCH_END</code>	This array holds the offsets of the ends of the last successful submatches in the currently active dynamic scope.
<code>%+</code>	Similar to <code>@+</code> , the <code>%+</code> hash allows access to the named capture buffers, should they exist, in the last successful match in the currently active dynamic scope.
<code>\$. , \$NR, \$INPUT_LINE_NUMBER</code>	The current line number for the last filehandle that was accessed.
<code>\$/ , \$RS, \$INPUT_RECORD_SEPARATOR</code>	The input record separator, newline by default.
<code>\$, \$OUTPUT_AUTOFLUSH</code>	If set to nonzero, this forces a flush right away and after every write or print on the currently selected output channel.
<code>\$. , \$OFS, \$OUTPUT_FIELD_SEPARATOR</code>	The output field separator for the print operator.
<code>\$\ , \$ORS, \$OUTPUT_RECORD_SEPARATOR</code>	The output record separator for the print operator.

VARIABLE	DESCRIPTION
<code>\$", \$LIST_SEPARATOR</code>	Like <code>\$,</code> except that it applies to array and slice values interpolated into a double-quoted string.
<code>\$;, \$SUBSEP, \$SUBSCRIPT_SEPARATOR</code>	The subscript separator for multidimensional array emulation.
<code>@-, @LAST_MATCH_START</code>	<code>\$-[0]</code> is the offset of the start of the last successful match. <code>\$-[n]</code> is the offset of the start of the substring matched by the <i>n</i> th subpattern, or undef if the subpattern did not match.
<code>%-</code>	Similar to <code>%+</code> , this variable allows access to the named capture buffers in the last successful match in the currently active dynamic scope.
<code>\$?, \$CHILD_ERROR</code>	The status returned by the last pipe close, backtick (``) command, successful call to <code>wait</code> or <code>waitpid</code> , or from the system operator.
<code>\${^CHILD_ERROR_NATIVE}</code>	The native status returned by the last pipe close, backtick (``) command, successful call to <code>wait</code> or <code>waitpid</code> , or from the system operator.
<code>\${^ENCODING}</code>	The object reference to the Encode module object that is used to convert the source code to Unicode.
<code>\$!, \$ERRNO, \$OS_ERROR</code>	If used numerically, it yields the current value of the C <code>errno</code> variable; in other words, if a system or library call fails, it sets this variable.
<code>%!, %ERRNO, %OS_ERROR</code>	Each element of <code>%!</code> has a true value only if <code>\$!</code> is set to that value.
<code>^E, \$EXTENDED_OS_ERROR</code>	Error information specific to the current operating system.
<code>\$@, \$EVAL_ERROR</code>	The Perl syntax error message from the last <code>eval()</code> operator.
<code>\$\$, \$PID, \$PROCESS_ID</code>	The process number of the Perl running this script.
<code>\$<, \$UID, \$REAL_USER_ID</code>	The real <code>uid</code> of this process.
<code>\$>, \$EUID, \$EFFECTIVE_USER_ID</code>	The effective <code>uid</code> of this process.
<code>\$ (, \$GID, \$REAL_GROUP_ID</code>	The real <code>gid</code> of this process.
<code>\$), \$EGID, \$EFFECTIVE_GROUP_ID</code>	The effective <code>gid</code> of this process.

(continued)

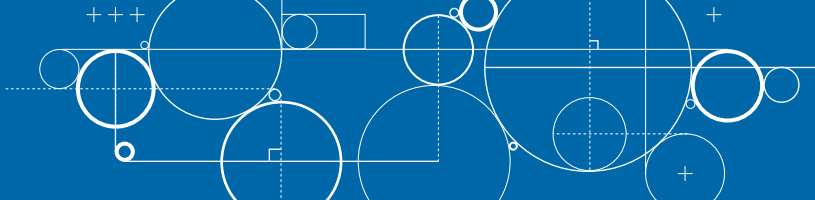
Using Perl Pre-Defined Variables (continued)



VARIABLE	DESCRIPTION
<code>\$0, \$PROGRAM_NAME</code>	Contains the name of the program being executed.
<code>\$[</code>	The index of the first element in an array, and of the first character in a substring.
<code>\$]</code>	The version + patchlevel / 1000 of the Perl interpreter.
<code>\$^C, \$COMPILING</code>	The current value of the flag associated with the <code>-c</code> switch.
<code>\$^D, \$DEBUGGING</code>	The current value of the debugging flags.
<code>\${^RE_DEBUG_FLAGS}</code>	The current value of the regular expression debugging flags.
<code>\${^RE_TRIE_MAXBUF}</code>	Controls how certain regular expression optimizations are applied and how much memory they utilize.
<code>\$^F, \$SYSTEM_FD_MAX</code>	The maximum system file descriptor, ordinarily 2.
<code>\$^I, \$INPLACE_EDIT</code>	The current value of the inplace-edit extension.
<code>\$^O, \$OSNAME</code>	The name of the operating system under which this copy of Perl was built, as determined during the configuration process.
<code>\$^S, \$EXCEPTIONS_BEING_CAUGHT</code>	The current state of the Perl interpreter.
<code>\$^T, \$BASETIME</code>	The time at which the program began running, in seconds since the epoch (beginning of 1970).
<code>\${^TAINT}</code>	Reflects if taint mode is on or off, representing whether variables can be influenced from events outside of your program.
<code>\${^UNICODE}</code>	Reflects certain Unicode settings of Perl.
<code>\${^UTF8CACHE}</code>	A variable that controls the state of the internal UTF-8 offset caching code.
<code>\${^UTF8LOCALE}</code>	A variable that indicates whether a UTF-8 locale was detected by Perl at startup.
<code>\$^V, \$PERL_VERSION</code>	The revision, version, and subversion of the Perl interpreter, represented as a version object.

VARIABLE	DESCRIPTION
<code>\$^W, \$WARNING</code>	The current value of the warning switch.
<code>\${^WARNING_BITS}</code>	The current set of warning checks enabled by the <code>use warnings</code> pragma.
<code>\${^WIN32_SLOPPY_STAT}</code>	If true, then <code>stat()</code> on Windows will not try to open the file.
<code>\$^X, \$EXECUTABLE_NAME</code>	The name used to execute the current copy of Perl.
<code>ARGV</code>	The special filehandle that iterates over command-line filenames in <code>@ARGV</code> .
<code>\$ARGV</code>	Contains the name of the current file when reading from <code><></code> .
<code>@ARGV</code>	Contains the command-line arguments intended for the script.
<code>ARGVOUT</code>	The special filehandle that points to the currently open output file when doing edit-in-place processing with <code>-i</code> .
<code>@F</code>	Contains the fields of each line read in when autosplit mode is turned on.
<code>@INC</code>	Contains the list of places where the <code>do</code> , <code>require</code> , or <code>use</code> constructs look for their library files.
<code>@_, @ARG</code>	Within a subroutine, the array <code>@_</code> contains the parameters passed to that subroutine.
<code>%INC</code>	Contains entries for each filename, included using the <code>do</code> , <code>require</code> , or <code>use</code> operators.
<code>%ENV</code>	Contains your current environment.
<code>%SIG</code>	Contains signal handlers for signals.

Perl Operators



Perl operators control variable assignment, testing, and manipulation. They exist as the special characters in between statements that allow you to state, calculate, append, search, replace, and compare

variables. You can access additional details and documentation with the PerlDoc program on the command-line using `perldoc perlop`.

Quote Operators

OPERATOR	APPLICATION	DESCRIPTION
" <i>STRING</i> "	Literal interpolate	Returns <i>STRING</i> , and resolve any Perl keywords that are found.
' <i>STRING</i> '	Literal non-interpolate	Returns the literal <i>STRING</i> .
` <i>COMMAND</i> `	Command Execute	Returns output of <i>COMMAND</i> as executed as a command-line program.
/ <i>EXPR</i> /	Pattern match	Performs a regular expression pattern match.

Quote-Like Operators

OPERATOR	DESCRIPTION
q/ <i>STRING</i> /	Returns <i>STRING</i> with literal single-quotes, like ' <i>STRING</i> '.
qq/ <i>STRING</i> /	Returns <i>STRING</i> with literal double-quotes, like " <i>STRING</i> ".
qw/ <i>STRING</i> /	Returns <i>STRING</i> split by words as an array.
qx/ <i>PROGRAM LIST</i> /	Executes a system command <i>PROGRAM</i> with arguments <i>LIST</i> , and returns the output of the program.

Regular Expression Quote-Like Operators

OPERATOR	DESCRIPTION
qr/ <i>STRING</i> /	Returns <i>STRING</i> as a regular expression pattern.
if (<i>VAR</i> =~ / <i>PATTERN</i> /) { ... }	True if <i>PATTERN</i> is found in <i>VAR</i> .
<i>VAR</i> =~ s/ <i>PATTERN</i> / <i>EXPR</i> / ;	Replaces <i>PATTERN</i> with <i>EXPR</i> in <i>VAR</i> .
if (<i>VAR</i> =~ s/ <i>PATTERN</i> / <i>EXPR</i> /) { ... }	True if <i>PATTERN</i> found in <i>VAR</i> , and replaced with <i>EXPR</i> .
<i>VAR</i> =~ tr/ <i>X</i> / <i>Y</i> / ;	Replaces a single character, <i>X</i> , with <i>Y</i> in <i>VAR</i> .

Mathematic Operators

OPERATOR	APPLICATION	DESCRIPTION
$NUM1 + NUM2$	Add	Returns $NUM1$ added to $NUM2$.
$NUM1 - NUM2$	Subtract	Returns $NUM2$ subtracted from $NUM1$.
$NUM1 * NUM2$	Multiply	Returns $NUM1$ multiplied by $NUM2$.
$NUM1 / NUM2$	Divide	Returns $NUM1$ divided by $NUM2$.
$NUM1 \% NUM2$	Modulus	Returns remainder of $NUM1$ divided by $NUM2$.
$NUM ** EXP$	Exponent	Multiplies NUM by itself EXP times.
$++VAR$	Pre-auto-increment	Increments VAR by one; returns new VAR value.
$--VAR$	Pre-auto-decrement	Decrements VAR by one; returns new VAR value.
$VAR++$	Post-auto-increment	Returns current VAR value; increments VAR by one.
$VAR--$	Post-auto-decrement	Returns current VAR value; decrements VAR by one.

Assignment Operators

Assignment operators exist as a way to store a value, on the right side of the statement, into a variable, on the left side. Some of these operators can do more than simply store a value into a variable; they can simultaneously manipulate the pre-existing variable's value, before it actually applies the new value.

Assignment Operator

OPERATOR	APPLICATION	DESCRIPTION
$VAR = VALUE$	Assign	Assign $VALUE$ into VAR .
$VAR =~ REGEXP$	Apply	Apply $REGEXP$ onto VAR .

Numeric Assignment Operators

OPERATOR	APPLICATION	DESCRIPTION
$VAR += NUM$	Add to	Add NUM to VAR .
$VAR -= NUM$	Subtract from	Subtract NUM from VAR .
$VAR *= NUM$	Multiply by	Multiply VAR by NUM .
$VAR /= NUM$	Divide by	Divide VAR by NUM .
$VAR **= NUM$	Exponent	Multiply VAR by itself NUM times.

String Assignment Operators

OPERATOR	APPLICATION	DESCRIPTION
$VAR .= STR$	Concatenate	Concatenate STR onto VAR .
$VAR \times= NUM$	Duplicate	Duplicate VAR by NUM times.

Perl Operators (continued)

Assignment Operators *(continued)*

Shift Assignment Operators

OPERATOR	APPLICATION	DESCRIPTION
<code>VAR <<= NUM</code>	Shift left	Shift <code>VAR</code> left by <code>NUM</code> bits.
<code>VAR >>= NUM</code>	Shift right	Shift <code>VAR</code> right by <code>NUM</code> bits.

Binary Test, or Equality Operators

Binary operators reduce a variable to a simple true or false status. Often, if a variable has a value status, its binary status is true. If the variable has not been defined, or if there is no value, it is false. You can use these binary operators within conditional tests.

OPERATOR	APPLICATION	DESCRIPTION
<code>! VAR</code>	Logical not	True if <code>VAR</code> is false, and false if <code>VAR</code> is true.
<code>VAR1 && VAR2</code>	Logical and	True if <code>VAR1</code> and <code>VAR2</code> are not zero.
<code>VAR1 VAR2</code>	Logical or	True if <code>VAR1</code> or <code>VAR2</code> is not zero.
<code>VAR1 // VAR2</code>	Logical defined-or	True if <code>VAR1</code> is defined, or <code>VAR2</code> is not zero.

Relational Operators

Relational operators are used in conditional tests. They provide a way of comparing the left side with the right side.

OPERATOR	APPLICATION	DESCRIPTION
<code>NUM1 == NUM2</code>	Numeric equal-to	True if <code>NUM1</code> is equal to <code>NUM2</code> .
<code>NUM1 != NUM2</code>	Numeric not-equal-to	True if <code>NUM1</code> is not equal to <code>NUM2</code> .
<code>NUM1 < NUM2</code>	Numeric less-than	True if <code>NUM1</code> is less than <code>NUM2</code> .
<code>NUM1 > NUM2</code>	Numeric greater-than	True if <code>NUM1</code> is greater than <code>NUM2</code> .
<code>NUM1 <= NUM2</code>	Numeric less-than or equal-to	True if <code>NUM1</code> is less than or equal to <code>NUM2</code> .
<code>NUM1 >= NUM2</code>	Numeric greater-than or equal-to	True if <code>NUM1</code> is greater than or equal to <code>NUM2</code> .
<code>VAR1 eq VAR2</code>	Binary equal-to	True if <code>VAR1</code> is equal to <code>VAR2</code> .
<code>VAR1 ne VAR2</code>	Binary not-equal-to	True if <code>VAR1</code> is not equal to <code>VAR2</code> .
<code>VAR1 lt VAR2</code>	Binary less-than	True if <code>VAR1</code> is less than <code>VAR2</code> .
<code>VAR1 gt VAR2</code>	Binary greater-than	True if <code>VAR1</code> is greater than <code>VAR2</code> .
<code>VAR1 le VAR2</code>	Binary less-than or equal-to	True if <code>VAR1</code> is less than or equal to <code>VAR2</code> .
<code>VAR1 ge VAR2</code>	Binary greater-than or equal-to	True if <code>VAR1</code> is greater than or equal to <code>VAR2</code> .

List Operator

A list operator is the opening and closing parenthesis around one or more variables or values. This makes it easier to deal with the data as an array. When you group multiple variables together as a list, without storing them into an array variable, they implicitly become an anonymous array.

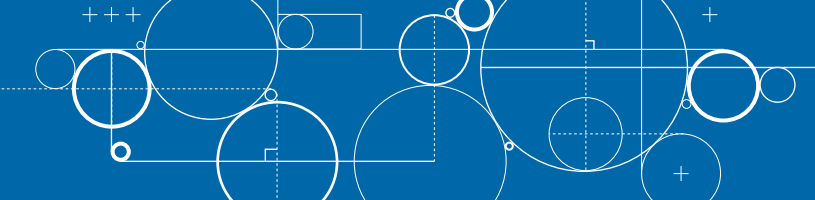
OPERATOR	APPLICATION	DESCRIPTION
(<i>VAR1</i> , <i>VAR2</i> , <i>VAR3</i> , ...)	Absolute list	Returns an array of <i>VAR1</i> , <i>VAR2</i> , <i>VAR3</i> .
(<i>VAR1</i> .. <i>VAR2</i>)	Binary range	Produces a list of all values between <i>VAR1</i> and <i>VAR2</i> .

Conditional Operator

A conditional operator is like combining an if-then-else conditional test into a single statement. There are always three parts, separated by a question mark and a colon. The first is the test, the second is the value used if the test is true, and the third is the value used if the test is false.

OPERATOR	APPLICATION	DESCRIPTION
<i>TEST</i> ? <i>VAR1</i> : <i>VAR2</i>	Conditional test statement	Returns a <i>VAR1</i> if <i>TEST</i> is true; otherwise, it returns a <i>VAR2</i> .

Perl Regular Expressions



A regular expression is a string that describes a pattern. In Perl, regular expressions are used for pattern matching for conditional tests, or for search-and-replace operations. A regular expression's capabilities are what give Perl its strength and versatility,

especially when it comes to pattern matching. Many other programming languages actually design their own regular expression syntax and functionality based on the Perl model.

You can access additional details and documentation with the `Perldoc` program on the command-line using the following commands:

```
perldoc perlrequick
perldoc perlretut
perldoc perlre
```

META-CHARACTER	DESCRIPTION
\	Quote the next meta-character.
^	Match the beginning of the line.
.	Match any character (except newline).
\$	Match the end of the line (or before newline).
	Define alternation.
()	Define grouping and capture buffer.
[xyz]	Define character class list of <i>x</i> , <i>y</i> , <i>z</i> .
[^xyz]	Define <i>not</i> character class list of <i>x</i> , <i>y</i> , <i>z</i> .
[a-z]	Define character class range of <i>a</i> to <i>z</i> .
[^a-z]	Define <i>not</i> character class range of <i>a</i> to <i>z</i> .

A character class represents a group of characters that share a common lexical characteristic. This allows you to specify a large number of characters by type, without specifying the literal characters in the pattern match.

CHARACTER CLASSES	DESCRIPTION
\w	Match a word character (alphanumeric plus "_").
\W	Match a non-word character.
\s	Match a whitespace character.
\S	Match a non-whitespace character.
\d	Match a digit character.
\D	Match a non-digit character.

A quantifier allows you to match by multiple copies of any character, meta-character, or character class in your pattern string. You can match the largest possible number of values in a string using *greedy quantifiers*; or the smallest possible number of values using *non-greedy quantifiers*.

GREEDY QUANTIFIERS	DESCRIPTION
•	Match zero or more times.
+	Match 1 or more times.
?	Match 1 or zero times.
{n}	Match exactly <i>n</i> times.
{n, }	Match at least <i>n</i> times.
{n,m}	Match at least <i>n</i> but not more than <i>m</i> times.

NON-GREEDY QUANTIFIERS	DESCRIPTION
*?	Match zero or more times, not greedily.
+?	Match 1 or more times, not greedily.
??	Match 1 or zero times, not greedily.
{n}?	Match exactly <i>n</i> times, not greedily.
{n, }?	Match at least <i>n</i> times, not greedily.
{n,m}?	Match at least <i>n</i> but not more than <i>m</i> times, not greedily.

A modifier flag allows you to change options controlling how entire regular expression pattern executes.

MODIFIER FLAGS	DESCRIPTION
m	Treat string as multiple lines.
s	Treat string as a single line.
e	Evaluate replacement text as Perl code.
i	Do case-insensitive pattern matching.
x	Extend pattern's legibility; allow whitespace and comments.
g	Enable global matching.
c	Keep current position after failed match.

Apache Run-Time Configuration Directives

Apache 2.2 supports almost 800 different configuration directives that allow you to customize its functionality as your Web server.

Each directive is context sensitive and can only be applied in specific locations in the Apache `httpd.conf` configuration file.

CONFIGURATION CONTEXT	DESCRIPTION
server	Specify and apply configuration directives globally to the Apache server.
virtual	Specify and apply configuration directives only within <code><virtualhost> ... </virtualhost></code> configuration groups.
directory	Specify and apply configuration directives only within <code><directory> ... </directory></code> configuration groups
.htaccess	Specify configuration directives only within <code>.htaccess</code> configuration files; apply only to a specific Web site directory containing the file.

If you modify anything in the server, virtual, or directory contexts, you must restart the Apache service to apply the change. If you apply your change to an `.htaccess` file, you do not need to restart the service.

You can find additional documentation on Apache directives online at <http://httpd.apache.org/docs/2.2/mod/directives.html>.

Core Apache Directives

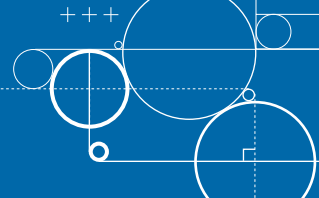
DIRECTIVE	DESCRIPTION	CONTEXT
<code>AcceptFilter</code> <i>protocol accept_filter</i>	Configures optimizations for a protocol's listener sockets	server
<code>AcceptPathInfo</code> On Off Default	Resources accept trailing pathname information	server, virtual, directory, .htaccess
<code>AccessFileName</code> <i>filename</i> [<i>filename</i>] ...	Sets the name of the distributed configuration file	server, virtual
<code>AddDefaultCharset</code> On Off <i>charset</i>	Sets the default charset parameter to be added when a response content-type is text/plain or text/html	server, virtual, directory, .htaccess
<code>AddOutputFilterByType</code> <i>filter</i> [: <i>filter</i>] <i>MIME-type</i> [<i>MIME-type</i>] ...	Assigns an output filter to a particular MIME-type	server, virtual, directory, .htaccess
<code>AllowEncodedSlashes</code> On Off	Determines whether encoded path separators in URLs are allowed to be passed through	server, virtual
<code>AllowOverride</code> All None <i>directive-type</i> [<i>directive-type</i>] ...	Types of directives that are allowed in <code>.htaccess</code> files	directory

DIRECTIVE	DESCRIPTION	CONTEXT
AuthName <i>auth-domain</i>	Sets the Authorization realm for use in HTTP authentication	directory, .htaccess
AuthType Basic Digest	Defines the type of user authentication supported	directory, .htaccess
CGIMapExtension <i>cgi-path .extension</i>	Configures a technique for locating the interpreter for CGI scripts	directory, .htaccess
ContentDigest On Off	Enables the generation of Content-MD5 HTTP Response headers	server, virtual, directory, .htaccess
DefaultType <i>MIME-type</i> None	Sets the MIME content-type that will be sent if the server cannot determine a type in any other way	server, virtual, directory, .htaccess
<Directory <i>directory-path</i> > ... </Directory>	Encloses a group of directives that apply only to the named file-system directory and subdirectories	server, virtual
<DirectoryMatch <i>regex</i> > ... </DirectoryMatch>	Encloses directives that apply to file-system directories matching a regular expression and their subdirectories	server, virtual
DocumentRoot <i>directory-path</i>	Sets a directory that forms the main document tree visible from the Web	server, virtual
EnableMMAP On Off	Uses memory-mapping to read files during delivery	server, virtual, directory, .htaccess
EnableSendfile On Off	Uses the kernel sendfile support to deliver files to the client	server, virtual, directory, .htaccess
ErrorDocument <i>error-code document</i>	Configures what the server will return to the client in case of an error	server, virtual, directory, .htaccess
ErrorLog <i>file-path</i> syslog[: <i>facility</i>]	Sets the location where the server will log errors	server, virtual
FileETag <i>component</i> ...	Configures file attributes used to create the ETag HTTP response header for static files	server, virtual, directory, .htaccess
<Files <i>filename</i> > ... </Files>	Contains directives that apply to matched filenames	server, virtual, directory, .htaccess
<FilesMatch <i>regex</i> > ... </FilesMatch>	Contains directives that apply to regular-expression matched filenames	server, virtual, directory, .htaccess
ForceType <i>MIME-type</i> None	Forces all matching files to be served with the specified MIME content-type	directory, .htaccess
HostnameLookups On Off Double	Enables DNS lookups on client IP addresses	server, virtual, directory
<IfDefine [!] <i>parameter-name</i> > ... </IfDefine>	Encloses directives that will be processed only if a test is true at startup	server, virtual, directory, .htaccess
<IfModule [!] <i>module-file</i> <i>module-identifier</i> > ... </IfModule>	Encloses directives that are processed conditional on the presence or absence of a specific module	server, virtual, directory, .htaccess
Include <i>file-path</i> <i>directory-path</i>	Includes other configuration files from within the server configuration files	server, virtual, directory

(continued)

continued

Apache Run-Time Configuration Directives (continued)



Core Apache Directives (continued)

DIRECTIVE	DESCRIPTION	CONTEXT
KeepAlive On Off	Enables HTTP persistent connections	server, virtual
KeepAliveTimeout <i>seconds</i>	Sets the amount of time the server will wait for subsequent requests on a persistent connection	server, virtual
<Limit <i>method</i> [<i>method</i>] ... > ... </Limit>	Restricts enclosed access controls to only certain HTTP methods	server, virtual, directory, .htaccess
<LimitExcept <i>method</i> [<i>method</i>] ... > ... </LimitExcept>	Restricts access controls to all HTTP methods except the named ones	server, virtual, directory, .htaccess
LimitInternalRecursion <i>number</i> [<i>number</i>]	Determines maximum number of internal redirects and nested subrequests	server, virtual
LimitRequestBody <i>bytes</i>	Restricts the total size of the HTTP request body sent from the client	server, virtual, directory, .htaccess
LimitRequestFields <i>number</i>	Limits the number of HTTP request header fields that will be accepted from the client	server
LimitRequestFieldSize <i>bytes</i>	Limits the size of the HTTP request header allowed from the client	server
LimitRequestLine <i>bytes</i>	Limits the size of the HTTP request line that will be accepted from the client	server
LimitXMLRequestBody <i>bytes</i>	Limits the size of an XML-based request body	server, virtual, directory, .htaccess
<Location <i>URL-path</i> <i>URL</i> > ... </Location>	Applies the enclosed directives only to matching URLs	server, virtual
<LocationMatch <i>regex</i> > ... </LocationMatch>	Applies the enclosed directives only to regular-expression matching URLs	server, virtual
LogLevel <i>level</i>	Controls the verbosity of the ErrorLog	server, virtual
MaxKeepAliveRequests <i>number</i>	Sets the number of requests allowed on a persistent connection	server, virtual
NameVirtualHost <i>addr</i> [: <i>port</i>]	Designates an IP address for name-virtual hosting	server
Options [+ -] <i>option</i> [[+ -] <i>option</i>] ...	Configures what features are available in a particular directory	server, virtual, directory, .htaccess
Require <i>entity-name</i> [<i>entity-name</i>] ...	Selects which authenticated users can access a resource	directory, .htaccess
RLimitCPU <i>seconds</i> max [<i>seconds</i> max]	Limits the CPU consumption of processes launched by Apache children	server, virtual, directory, .htaccess
RLimitMEM <i>bytes</i> max [<i>bytes</i> max]	Limits the memory consumption of processes launched by Apache children	server, virtual, directory, .htaccess
RLimitNPROC <i>number</i> max [<i>number</i> max]	Limits the number of processes that can be launched by processes launched by Apache children	server, virtual, directory, .htaccess

Core Apache Directives (continued)

DIRECTIVE	DESCRIPTION	CONTEXT
Satisfy Any All	Interaction between host-level access control and user authentication	directory, .htaccess
ScriptInterpreterSource Registry Registry-Strict Script	Configures the technique for locating the interpreter for CGI scripts	server, virtual, directory, .htaccess
ServerAdmin <i>email-address</i> <i>URL</i>	Sets the e-mail address that the server includes in error messages sent to the client	server, virtual
ServerAlias <i>hostname</i> [<i>hostname</i>] ...	Configures alternate names for a host used when matching requests to name-virtual hosts	virtual
ServerName [<i>scheme://</i>] <i>fully-qualified-domain-name</i> [: <i>port</i>]	Configures the hostname and port that the server uses to identify itself	server, virtual
ServerPath <i>URL-path</i>	Sets the legacy URL pathname for a name-based virtual host that is accessed by an incompatible browser	virtual
ServerRoot <i>directory-path</i>	Configures the base directory for the server installation	server
ServerSignature On Off EMail	Configures the footer on server-generated documents	server, virtual, directory, .htaccess
ServerTokens Major Minor Min[imal] Prod[uctOnly] OS Full	Configures the Server HTTP response header	server
SetHandler <i>handler-name</i> None	Forces all matching files to be processed by a handler	server, virtual, directory, .htaccess
SetInputFilter <i>filter</i> [: <i>filter</i> ...]	Sets the filters that will process client requests and POST input	server, virtual, directory, .htaccess
SetOutputFilter <i>filter</i> [: <i>filter</i> ...]	Sets the filters that will process responses from the server	server, virtual, directory, .htaccess
TimeOut <i>seconds</i>	Amount of time the server will wait for certain events before failing a request	server, virtual
TraceEnable [On Off extended]	Determines the behavior on TRACE requests	server
UseCanonicalName On Off DNS	Configures how the server determines its own name and port	server, virtual, directory
UseCanonicalPhysicalPort On Off	Configures how the server determines its own name and port	server, virtual, directory
Win32DisableAcceptEx	Uses <code>accept()</code> rather than <code>AcceptEx()</code> to accept network connections	server

Apache Run-Time Configuration Directives (continued)

Multi-Process Module Directives

Multi-Process Module (MPM) directives are responsible for binding to network ports on the server, accepting requests, and dispatching children to handle the requests.

DIRECTIVE	DESCRIPTION	CONTEXT
AcceptMutex <i>Default method</i>	Configures the method that Apache uses to serialize multiple children accepting requests on network sockets	server
Group <i>unix-group</i>	Sets the group under which the server will answer requests	server
Listen [<i>IP-address</i> :] <i>portnumber</i> [<i>protocol</i>]	Sets the IP addresses and ports that the server listens to	server
ListenBacklog <i>backlog</i>	Sets the maximum length of the queue of pending connections	server
LockFile <i>filename</i>	Sets the location of the serialization lock file	server
MaxClients <i>number</i>	Sets the maximum number of connections that will be processed simultaneously	server
MaxMemFree <i>KBytes</i>	Sets the maximum amount of memory that the main allocator is allowed to hold without calling <code>free()</code>	server
MaxRequestsPerChild <i>number</i>	Sets the maximum number of requests that an individual child server will handle during its life	server
MaxSpareServers <i>number</i>	Sets the maximum number of idle child server processes	server
MaxSpareThreads <i>number</i>	Sets the maximum number of idle threads	server
MinSpareServers <i>number</i>	Sets the minimum number of idle child server processes	server
MinSpareThreads <i>number</i>	Sets the minimum number of idle threads available to handle request spikes	server
PidFile <i>filename</i>	Sets the file where the server records the process ID of the daemon	server
ReceiveBufferSize <i>bytes</i>	Sets the size of the TCP receive buffer	server
ScoreBoardFile <i>file-path</i>	Sets the location of the file used to store coordination data for the child processes	server
SendBufferSize <i>bytes</i>	Sets the TCP buffer size	server
ServerLimit <i>number</i>	Sets the upper limit on configurable number of processes	server
StartServers <i>number</i>	Sets the number of child server processes created at startup	server
ThreadLimit <i>number</i>	Sets the upper limit on the configurable number of threads per child process	server
ThreadsPerChild <i>number</i>	Sets the number of threads created by each child process	server
ThreadStackSize <i>size</i>	Sets the size in bytes of the stack used by threads handling client connections	server
User <i>unix-userid</i>	Sets the local userID under which the server will answer requests	server

Apache Base Modules and Directives

Below is a list of all of the base modules that come as part of the Apache distribution. These are considered base modules because they allow Apache to act as a standard Web server.

mod_actions

The `mod_actions` module has two directives. The `Action` directive lets you run CGI scripts whenever a file of a certain MIME content type is requested. The `Script` directive lets you run CGI scripts whenever a particular method is used in a request. This makes it much easier to execute scripts that process files.

DIRECTIVE	DESCRIPTION	CONTEXT
Action <i>action-type cgi-script</i> [<i>virtual</i>]	Activates a CGI script for a particular handler or content-type	server, virtual, directory, .htaccess
Script <i>method cgi-script</i>	Activates a CGI script for a particular request method	server, virtual, directory

mod_alias

The directives contained in the `mod_alias` module allow for manipulation and control of URLs as requests arrive at the server. The `Alias` and `ScriptAlias` directives are used to map between URLs and filesystem paths. This allows for content that is not directly under the `DocumentRoot`

configuration path to be served as part of the Web site from an aliased URL path. The `ScriptAlias` directive has the additional effect of marking the target directory as containing only CGI scripts.

DIRECTIVE	DESCRIPTION	CONTEXT
Alias <i>URL-path file-path</i> <i>directory-path</i>	Maps URLs to filesystem locations	server, virtual
AliasMatch <i>regex file-path</i> <i>directory-path</i>	Maps URLs to filesystem locations using regular expressions	server, virtual
Redirect [<i>status</i>] <i>URL-path URL</i>	Sends an external redirect asking the client to fetch a different URL	server, virtual, directory, .htaccess
RedirectMatch [<i>status</i>] <i>regex URL</i>	Sends an external redirect based on a regular expression match of the current URL	server, virtual, directory, .htaccess
RedirectPermanent <i>URL-path URL</i>	Sends an external permanent redirect asking the client to fetch a different URL	server, virtual, directory, .htaccess
RedirectTemp <i>URL-path URL</i>	Sends an external temporary redirect asking the client to fetch a different URL	server, virtual, directory, .htaccess
ScriptAlias <i>URL-path file-path</i> <i>directory-path</i>	Maps a URL to a filesystem location and designates the target as a CGI script	server, virtual
ScriptAliasMatch <i>regex file-path</i> <i>directory-path</i>	Maps a URL to a filesystem location using a regular expression and designates the target as a CGI script	server, virtual

Apache Base Modules and Directives (continued)

mod_autoindex

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The `DirectoryIndex` directive sets the name of this file. This is controlled by `mod_dir`.
- A listing generated by the server. The other directives control the format of this listing. The `AddIcon`, `AddIconByEncoding`, and `AddIconByType` directives are used to set a list of icons to display for various file types; for each

file listed, the first icon listed that matches the file is displayed. These are controlled by `mod_autoindex`.

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

Automatic index generation is enabled when using the “Indexes” option. See the `Options` directive for more details.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AddAlt string file [file] ...</code>	Assigns the alternate text to display for a file, instead of an icon selected by filename	server, virtual, directory, <code>.htaccess</code>
<code>AddAltByEncoding string MIME-encoding [MIME-encoding] ...</code>	Assigns the alternate text to display for a file instead of an icon selected by MIME-encoding	server, virtual, directory, <code>.htaccess</code>
<code>AddAltByType string MIME-type [MIME-type] ...</code>	Assigns the alternate text to display for a file, instead of an icon selected by MIME content-type	server, virtual, directory, <code>.htaccess</code>
<code>AddDescription string file [file] ...</code>	Assigns a description to display for a file	server, virtual, directory, <code>.htaccess</code>
<code>AddIcon icon name [name] ...</code>	Assigns an icon to display for a file selected by name	server, virtual, directory, <code>.htaccess</code>
<code>AddIconByEncoding icon MIME-encoding [MIME-encoding] ...</code>	Assigns an icon to display next to files selected by MIME content-encoding	server, virtual, directory, <code>.htaccess</code>
<code>AddIconByType icon MIME-type [MIME-type] ...</code>	Assigns an icon to display next to files selected by MIME content-type	server, virtual, directory, <code>.htaccess</code>
<code>DefaultIcon url-path</code>	Sets the default icon to display for files when no specific icon is configured	server, virtual, directory, <code>.htaccess</code>
<code>HeaderName filename</code>	Configures the name of the file that will be inserted at the top of the index listing	server, virtual, directory, <code>.htaccess</code>
<code>IndexHeadInsert “markup ...”</code>	Inserts text in the HEAD section of an index page	server, virtual, directory, <code>.htaccess</code>
<code>IndexIgnore file [file] ...</code>	Adds to the list of files to hide when listing a directory	server, virtual, directory, <code>.htaccess</code>
<code>IndexOptions [+ -]option [[+ -]option] ...</code>	Configures various settings for directory indexing	server, virtual, directory, <code>.htaccess</code>
<code>IndexOrderDefault Ascending Descending Name Date Size Description</code>	Sets the default ordering of the directory index	server, virtual, directory, <code>.htaccess</code>
<code>IndexStyleSheet url-path</code>	Adds a CSS stylesheet to the directory index	server, virtual, directory, <code>.htaccess</code>
<code>ReadmeName filename</code>	Configures the name of the file that will be inserted at the end of the index listing	server, virtual, directory, <code>.htaccess</code>

mod_cgi

Any file that has the handler `cgi-script` will be treated as a CGI script, and run by the server, with its output being returned to the client. Files acquire this handler either by

having a name containing an extension defined by the `AddHandler` directive, or by being in a `ScriptAlias` directory.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>ScriptLog file-path</code>	Sets the location of the CGI script error logfile	server, virtual
<code>ScriptLogBuffer bytes</code>	Sets the maximum amount of PUT or POST requests that will be recorded in the scriptlog	server, virtual
<code>ScriptLogLength bytes</code>	Sets the size limit of the CGI script logfile	server, virtual

mod_dir

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The `DirectoryIndex` directive sets the name of this file. This is controlled by `mod_dir`.

- A listing generated by the server. This is provided by `mod_autoindex`.

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>DirectoryIndex local-URL [local-URL] ...</code>	Configures a list of resources to look for when the client requests a directory	server, virtual, directory, <code>.htaccess</code>
<code>DirectorySlash On Off</code>	Toggles trailing slash redirects on or off	server, virtual, directory, <code>.htaccess</code>

mod_env

The `mod_env` module allows for control of the environment that will be provided to CGI scripts and SSI pages. Environment variables may be passed from the shell that

invoked the `httpd` process. Alternatively, environment variables may be set or unset within the configuration process.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>PassEnv env-variable [env-variable] ...</code>	Passes environment variables from the shell	server, virtual, directory, <code>.htaccess</code>
<code>SetEnv env-variable value</code>	Sets environment variables	server, virtual, directory, <code>.htaccess</code>
<code>UnsetEnv env-variable [env-variable] ...</code>	Removes variables from the environment	server, virtual, directory, <code>.htaccess</code>

Apache Base Modules and Directives (continued)

mod_filter

The `mod_filter` module enables smart, context-sensitive configuration of output content filters. For example, you can configure Apache to process different content-types through different filters, even when you do not know the content-type in advance (for example, in a proxy).

DIRECTIVE	DESCRIPTION	CONTEXT
<code>FilterChain [+!@]filter-name ...</code>	Configures the filter chain	server, virtual, directory, .htaccess
<code>FilterDeclare filter-name [type]</code>	Declares a smart filter	server, virtual, directory, .htaccess
<code>FilterProtocol filter-name [provider-name] proto-flags</code>	Deals with correct HTTP protocol handling	server, virtual, directory, .htaccess
<code>FilterProvider filter-name provider-name [req resp env]=dispatch match</code>	Registers a content filter <code>mod_filter</code>	server, virtual, directory, .htaccess

mod_imagemap

The `mod_imagemap` module allows for control of the environment that will be provided to CGI scripts and SSI pages. Environment variables may be passed from the shell that invoked the `httpd` process. Alternatively, environment variables may be set or unset within the configuration process.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>ImapBase map referer URL</code>	Configures the default base for imagemap files	server, virtual, directory, .htaccess
<code>ImapDefault error nocontent map referer URL</code>	Configures the default action when an imagemap is called with coordinates that are not explicitly mapped	server, virtual, directory, .htaccess
<code>ImapMenu none formatted semiformatted unformatted</code>	Configures the action if no coordinates are given when calling an imagemap	server, virtual, directory, .htaccess

mod_log_config

The `mod_log_config` module allows for control of the environment that will be provided to CGI scripts and SSI pages. Environment variables may be passed from the shell that invoked the `httpd` process. Alternatively, environment variables may be set or unset within the configuration process.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>BufferedLogs On Off</code>	Buffers log entries in memory before writing to disk	server
<code>CookieLog filename</code>	Sets a filename for the logging of cookies	server, virtual
<code>CustomLog file pipe format nickname [env=[!] environment-variable]</code>	Sets the filename and format of a log file	server, virtual
<code>LogFormat format nickname [nickname]</code>	Describes a format for use in a log file	server, virtual
<code>TransferLog file pipe</code>	Specifies the location of a log file	server, virtual

mod_mime

The `mod_mime` module allows for control of the environment that will be provided to CGI scripts and SSI pages. Environment variables may be passed from the shell that invoked the `httpd` process. Alternatively, environment variables may be set or unset within the configuration process.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AddCharset</code> <i>charset extension</i> [<i>extension</i>] ...	Maps the given filename extensions to the specified content charset	server, virtual, directory, <code>.htaccess</code>
<code>AddEncoding</code> <i>MIME-encoding extension</i> [<i>extension</i>] ...	Maps the given filename extensions to the specified encoding type	server, virtual, directory, <code>.htaccess</code>
<code>AddHandler</code> <i>handler-name extension</i> [<i>extension</i>] ...	Maps the filename extensions to the specified handler	server, virtual, directory, <code>.htaccess</code>
<code>AddInputFilter</code> <i>filter[:filter] extension</i> [<i>extension</i>] ...	Maps filename extensions to the filters that will process client requests	server, virtual, directory, <code>.htaccess</code>
<code>AddLanguage</code> <i>MIME-lang extension</i> [<i>extension</i>] ...	Maps the given filename extension to the specified content language	server, virtual, directory, <code>.htaccess</code>
<code>AddOutputFilter</code> <i>filter[:filter] extension</i> [<i>extension</i>] ...	Maps filename extensions to the filters that will process responses from the server	server, virtual, directory, <code>.htaccess</code>
<code>AddType</code> <i>MIME-type extension</i> [<i>extension</i>] ...	Maps the given filename extensions onto the specified content type	server, virtual, directory, <code>.htaccess</code>
<code>DefaultLanguage</code> <i>MIME-lang</i>	Sets all files in the given scope to the specified language	server, virtual, directory, <code>.htaccess</code>
<code>ModMimeUsePathInfo</code> On Off	Tells <code>mod_mime</code> to treat <code>path_info</code> components as part of the filename	directory
<code>MultiviewsMatch</code> Any NegotiatedOnly [Filters] Handlers [Handlers] Filters]	Configures the types of files that will be included when searching for a matching file with MultiViews	server, virtual, directory, <code>.htaccess</code>
<code>RemoveCharset</code> <i>extension</i> [<i>extension</i>] ...	Removes any character set associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveEncoding</code> <i>extension</i> [<i>extension</i>] ...	Removes any content encoding associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveHandler</code> <i>extension</i> [<i>extension</i>] ...	Removes any handler associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveInputFilter</code> <i>extension</i> [<i>extension</i>] ...	Removes any input filter associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveLanguage</code> <i>extension</i> [<i>extension</i>] ...	Removes any language associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveOutputFilter</code> <i>extension</i> [<i>extension</i>] ...	Removes any output filter associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>RemoveType</code> <i>extension</i> [<i>extension</i>] ...	Removes any content type associations for a set of file extensions	virtual, directory, <code>.htaccess</code>
<code>TypesConfig</code> <i>file-path</i>	Configures the location of the <code>mime.types</code> file	server

continued →

Apache Base Modules and Directives (continued)

mod_negotiation

Content negotiation, or more accurately, content selection, is the selection of the document that best matches the client's capabilities, from one of several available documents. There are two implementations of this:

- A type map (a file with the handler type-map) that explicitly lists the files containing the variants.
- A MultiViews search (enabled by the MultiViews Options), where the server does an implicit filename pattern match, and chooses from among the results.

DIRECTIVE	DESCRIPTION	CONTEXT
CacheNegotiatedDocs On Off	Allows content-negotiated documents to be cached by proxy servers	server, virtual
ForceLanguagePriority None Prefer Fallback [Prefer Fallback]	Configures the action to take if a single acceptable document is not found	server, virtual, directory, .htaccess
LanguagePriority <i>MIME-lang</i> [<i>MIME-lang</i>] ...	Sets the precedence of language variants for cases where the client does not express a preference	server, virtual, directory, .htaccess

mod_setenvif

The `mod_setenvif` module allows you to set environment variables according to whether different aspects of the request match regular expressions you specify. Other parts of the server

can use these environment variables to make decisions about actions to be taken.

DIRECTIVE	DESCRIPTION	CONTEXT
BrowserMatch <i>regex</i> [!]env-variable[=value] [[!]env-variable[=value]] ...	Sets environment variables conditional on HTTP User-Agent	server, virtual, directory, .htaccess
BrowserMatchNoCase <i>regex</i> [!]env-variable[=value] [[!]env-variable[=value]] ...	Sets environment variables conditional on User-Agent without respect to case	server, virtual, directory, .htaccess
SetEnvIf <i>attribute regex</i> [!]env-variable[=value] [[!]env-variable[=value]] ...	Sets environment variables based on attributes of the request	server, virtual, directory, .htaccess
SetEnvIfNoCase <i>attribute regex</i> [!]env-variable[=value] [[!]env-variable[=value]] ...	Sets environment variables based on attributes of the request without respect to case	server, virtual, directory, .htaccess

mod_status

The `mod_status` module allows a server administrator to find out how well their server is performing. An HTML page is presented that gives the current server statistics in an easily readable form. If required, this page can be made to

automatically refresh (if the browser supports refreshing). Another page gives a simple machine-readable list of the current server state.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>ExtendedStatus On Off</code>	Keeps track of extended status information for each request	server
<code>SeeRequestTail On Off</code>	Determines if <code>mod_status</code> displays the first 63 characters of a request or the last 63 characters, assuming the request is greater than 63 characters	server

mod_userdir

The `mod_userdir` module allows user-specific directories to be accessed using the `http://example.com/~user/` syntax.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>UserDir <i>directory-filename</i> [<i>directory-filename</i>] ...</code>	Sets the location of the user-specific directories	server, virtual

Apache Authentication and Authorization Modules and Directives

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone

is allowed to be where they want to go, or to have information that they want to have.

mod_auth_basic

The `mod_auth_basic` module allows the use of HTTP Basic Authentication to restrict access by looking up users in the given providers. HTTP Digest Authentication is provided by `mod_auth_digest`. You should usually combine this module with at least one authentication module such as `mod_authn_file` and one authorization module such as `mod_authz_user`.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthBasicAuthoritative On Off</code>	Sets whether authorization and authentication are passed to lower-level modules	directory, <code>.htaccess</code>
<code>AuthBasicProvider provider-name [provider-name] ...</code>	Sets the authentication provider(s) for this location	directory, <code>.htaccess</code>

mod_auth_digest

The `mod_auth_digest` module implements HTTP Digest Authentication (RFC 2617), and provides a more secure alternative to `mod_auth_basic`.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthDigestAlgorithm MD5 MD5-sess</code>	Selects the algorithm used to calculate the challenge and response hashes in digest authentication	directory, <code>.htaccess</code>
<code>AuthDigestDomain URI [URI] ...</code>	Configures the URIs that are in the same protection space for digest authentication	directory, <code>.htaccess</code>
<code>AuthDigestNcCheck On Off</code>	Enables or disables checking of the nonce-count sent by the server	server
<code>AuthDigestNonceFormat format</code>	Determines how the nonce is generated	directory, <code>.htaccess</code>
<code>AuthDigestNonceLifetime seconds</code>	Sets how long the server nonce is valid	directory, <code>.htaccess</code>
<code>AuthDigestProvider provider-name [provider-name] ...</code>	Sets the authentication provider(s) for this location	directory, <code>.htaccess</code>
<code>AuthDigestQop none auth auth-int [auth auth-int]</code>	Determines the quality-of-protection to use in digest authentication	directory, <code>.htaccess</code>
<code>AuthDigestShmemSize size</code>	Sets the amount of shared memory to allocate for keeping track of clients	server

mod_auth_alias

The `mod_auth_alias` module allows extended authentication providers to be created within the configuration file and assigned an alias name. The alias providers can then be referenced through the directives `AuthBasicProvider` or `AuthDigestProvider` in the

same way as a base authentication provider. Besides the ability to create and alias an extended provider, it also allows the same extended authentication provider to be referenced by multiple locations.

DIRECTIVE	DESCRIPTION	CONTEXT
<code><AuthnProviderAlias <i>baseProvider Alias</i>></code> ... <code></AuthnProviderAlias></code>	Encloses a group of directives that represent an extension of a base authentication provider and referenced by the <i>Alias</i> URL	server

mod_auth_anon

The `mod_auth_anon` module provides authentication front-ends such as `mod_auth_basic` to authenticate users similar to anonymous FTP sites — that is, they have a

userID *anonymous* and the e-mail address as a password. These e-mail addresses can be logged.

DIRECTIVE	DESCRIPTION	CONTEXT
Anonymous user [<i>user</i>] ...	Specifies userIDs that are allowed access without password verification	directory, .htaccess
Anonymous_LogEmail On Off	Specifies whether the password entered will be logged in the error log	directory, .htaccess
Anonymous_MustGiveEmail On Off	Specifies whether blank passwords are allowed	directory, .htaccess
Anonymous_NoUserID On Off	Specifies whether the userID field may be empty	directory, .htaccess
Anonymous_VerifyEmail On Off	Specifies whether to check the password field for a correctly formatted e-mail address	directory, .htaccess

mod_auth_dbd

The `mod_auth_dbd` module provides authentication front-ends such as `mod_auth_digest` and `mod_auth_basic` to authenticate users by looking them up in SQL tables. Modules such as `mod_auth_file` provide similar functionality.

This module relies on `mod_dbd` to specify the back-end database driver and connection parameters, and manage the database connections.

When using `mod_auth_basic` or `mod_auth_digest`, this module is invoked using the `AuthBasicProvider` or `AuthDigestProvider` with the `dbd` value.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthDBDUserPWQuery <i>query</i></code>	Configures the SQL query to look up a password for a user	directory
<code>AuthDBDUserRealmQuery <i>query</i></code>	Configures the SQL query to look up a password hash for a user and realm	directory

Apache Authentication and Authorization Modules and Directives (continued)

mod_authn_dbm

The `mod_authn_dbm` module provides authentication front-ends such as `mod_auth_digest` and `mod_auth_basic` to authenticate users by looking them up in `dbm` password files. The `mod_authn_file` module provides similar functionality.

When using `mod_auth_basic` or `mod_auth_digest`, this module is invoked using the `AuthBasicProvider` or `AuthDigestProvider` with the `dbm` value.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthDBMType</code> default SDBM GDBM NDBM DB	Sets the type of database file that is used to store passwords	directory, <code>.htaccess</code>
<code>AuthDBMUserFile</code> <i>file-path</i>	Sets the name of a database file containing the list of users and passwords for authentication	directory, <code>.htaccess</code>

mod_authn_default

The `mod_authn_default` module is designed to be the fallback module, if you have not configured an authentication module such as `mod_auth_basic`. It simply rejects any credentials supplied by the user.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthDefaultAuthoritative</code> On Off	Specifies whether authentication is passed to lower-level modules	directory, <code>.htaccess</code>

mod_authn_file

The `mod_authn_file` module provides authentication front-ends such as `mod_auth_digest` and `mod_auth_basic` to authenticate users by looking them up in plain-text password files. The `mod_authn_dbm` module provides similar functionality.

When using `mod_auth_basic` or `mod_auth_digest`, this module is invoked using the `AuthBasicProvider` or `AuthDigestProvider` with the `file` value.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthUserFile</code> <i>file-path</i>	Sets the name of a text file containing the list of users and passwords for authentication	directory, <code>.htaccess</code>

mod_authnz_ldap

The `mod_authnz_ldap` module provides authentication front-ends such as `mod_auth_basic` to authenticate users through an `ldap` directory.

`mod_authnz_ldap` supports the following features:

- It is known to support the OpenLDAP SDK (both 1.x and 2.x), Novell LDAP SDK, and the iPlanet (Netscape) SDK.
- Complex authorization policies can be implemented by representing the policy with LDAP filters.
- Uses extensive caching of LDAP operations through `mod_ldap`.
- Support for LDAP over SSL (requires the Netscape SDK) or TLS (requires the OpenLDAP 2.x SDK or Novell LDAP SDK).

When using `mod_auth_basic`, this module is invoked using the `AuthBasicProvider` directive with the `ldap` value.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthLDAPBindAuthoritative On Off</code>	Determines if other authentication providers are used when a user can be mapped to a DN but the server cannot successfully bind with the user's credentials	directory, <code>.htaccess</code>
<code>AuthLDAPBindDN distinguished-name</code>	Sets the optional DN to use in binding to the LDAP server	directory, <code>.htaccess</code>
<code>AuthLDAPBindPassword password</code>	Sets the password used in conjunction with the bind DN	directory, <code>.htaccess</code>
<code>AuthLDAPCharsetConfig file-path</code>	Sets the language-to-charset conversion configuration file	server
<code>AuthLDAPCompareDNOnServer On Off</code>	Allow the LDAP server to compare the DNs	directory, <code>.htaccess</code>
<code>AuthLDAPDereferenceAliases never searching finding always</code>	Specifies when the module de-references aliases	directory, <code>.htaccess</code>
<code>AuthLDAPGroupAttribute attribute</code>	Configures the LDAP attributes used to check for group membership	directory, <code>.htaccess</code>
<code>AuthLDAPGroupAttributeIsDN On Off</code>	Enables whether the DN of the client username is used when checking for group membership	directory, <code>.htaccess</code>
<code>AuthLDAPRemoteUserAttribute uid</code>	Configure the value of the attribute returned during the user query to set the <code>REMOTE_USER</code> environment variable	directory, <code>.htaccess</code>
<code>AuthLDAPRemoteUserIsDN On Off</code>	Enables whether the DN of the client username is used to set the <code>REMOTE_USER</code> environment variable	directory, <code>.htaccess</code>
<code>AuthLDAPUrl url [NONE SSL TLS STARTTLS]</code>	URL specifying the LDAP search parameters	directory, <code>.htaccess</code>
<code>AuthzLDAPAuthoritative On Off</code>	Prevents other authentication modules from authenticating the user if this one fails	directory, <code>.htaccess</code>

Apache Authentication and Authorization Modules and Directives (continued)

mod_authz_dbm

The `mod_authz_dbm` module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the Web site by group membership.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthDBMGroupFile</code> <i>file-path</i>	Sets the name of the database file containing the list of user groups for authorization	directory, <code>.htaccess</code>
<code>AuthzDBMAuthoritative</code> On Off	Sets whether authorization will be passed on to lower-level modules	directory, <code>.htaccess</code>
<code>AuthzDBMType</code> default SDBM GDBM NDBM DB	Sets the type of database file that is used to store a list of user groups	directory, <code>.htaccess</code>

mod_authz_default

The `mod_authz_default` module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the Web site by group membership.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthzDefaultAuthoritative</code> On Off	Sets whether authorization is passed to lower-level modules	directory, <code>.htaccess</code>

mod_authz_groupfile

The `mod_authz_groupfile` module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the Web site by group membership.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthGroupFile</code> <i>file-path</i>	Sets the name of a text file containing the list of user groups for authorization	directory, <code>.htaccess</code>
<code>AuthzGroupFileAuthoritative</code> On Off	Sets whether authorization will be passed on to lower-level modules	directory, <code>.htaccess</code>

mod_auth_host

The directives provided by `mod_auth_host` are used in `<Directory>`, `<Files>`, and `<Location>` sections as well as `.htaccess` files to control access to particular parts of the server. You can control access based on the client hostname, IP address, or other characteristics of the client request, as captured in environment variables. You use the `Allow` and `Deny` directives to specify which clients are or are not allowed access to the server, while you use the `Order` directive to set the default access state, and configure how the `Allow` and `Deny` directives interact with each other.

Both host-based access restrictions and password-based authentication may be implemented simultaneously. In that

case, you use the `Satisfy` directive to determine how the two sets of restrictions interact. The `Satisfy` directive has two possible configurations: `Satisfy Any` requires at least one restriction to pass, `Satisfy All` requires all restrictions to pass.

In general, access restriction directives apply to all access methods (GET, PUT, POST, and so on). This is the desired behavior in most cases. However, it is possible to restrict some methods, while leaving other methods unrestricted, by enclosing the directives in a `<Limit>` section.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>Allow from all</code> <code>host</code> <code>env=[!]<i>env-variable</i></code> <code>[host env=[!]<i>env-variable</i>] ...</code>	Controls which hosts can access an area of the server	directory, <code>.htaccess</code>
<code>Deny from all</code> <code>host</code> <code>env=[!]<i>env-variable</i></code> <code>[host env=[!]<i>env-variable</i>] ...</code>	Controls which hosts are denied access to the server	directory, <code>.htaccess</code>
<code>Order <i>ordering</i></code>	Controls the default access state and the order in which the <code>Allow</code> and <code>Deny</code> directives are evaluated	directory, <code>.htaccess</code>

mod_auth_owner

The `mod_auth_owner` module authorizes access to files by comparing the `userID` used for HTTP authentication (the Web `userID`) with the file-system owner or group of the

requested file. The supplied username and password must already be properly verified by an authentication module, such as `mod_auth_basic` or `mod_auth_digest`.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthzOwnerAuthoritative On Off</code>	Sets whether authorization will be passed on to lower-level modules	directory, <code>.htaccess</code>

mod_auth_user

The `mod_auth_user` module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the Web site. The `mod_auth_user` module grants access if the authenticated

user is listed in a `Require user` directive. Alternatively, you can use `Require valid-user` to grant access to all successfully authenticated users.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AuthzUserAuthoritative On Off</code>	Sets whether authorization will be passed on to lower-level modules	directory, <code>.htaccess</code>

Apache Extended Modules and Directives

Below is a list of extended modules that come as part of the Apache distribution. They are not a part of the base group because they provide optional, extended functionality beyond a normal Web server.

mod_cache

mod_cache module implements an RFC 2616-compliant HTTP content cache that you can use to cache either local or proxied content. This module requires the services of one or more storage management modules.

DIRECTIVE	DESCRIPTION	CONTEXT
CacheDefaultExpire <i>seconds</i>	Configures the default duration to cache a document when no expiry date is specified	server, virtual
CacheDisable <i>url-string</i>	Disables caching of specified URLs	server, virtual
CacheEnable <i>cache_type url-string</i>	Enables caching of specified URLs using a specified storage manager	server, virtual
CacheIgnoreCacheControl On Off	Ignores a request to not serve cached content to a client	server, virtual
CacheIgnoreHeaders <i>header-string [header-string] ...</i>	Does not allow the given HTTP header(s) to be stored in the cache	server, virtual
CacheIgnoreNoLastMod On Off	Ignores the fact that a response has no Last Modified header	server, virtual
CacheIgnoreQueryString On Off	Ignores a query string when caching	server, virtual
CacheIgnoreURLSessionIdentifiers <i>identifier [identifier] ...</i>	Ignores defined session identifiers encoded in the URL when caching	server, virtual
CacheLastModifiedFactor <i>float</i>	Sets the factor used to compute an expiry date based on the LastModified date	server, virtual
CacheLock On Off	Enables the mod_cache locking functionality	server, virtual
CacheLockMaxAge <i>integer</i>	Sets the maximum possible age of a cache lock	server, virtual
CacheLockPath <i>directory</i>	Sets the lock path directory	server, virtual
CacheMaxExpire <i>seconds</i>	Sets the maximum time in seconds to cache a document	server, virtual
CacheStoreNoStore On Off	Attempts to cache requests or responses that have been marked as no-store	server, virtual
CacheStorePrivate On Off	Attempts to cache responses that the server has marked as private	server, virtual

mod_cern_meta

The `mod_cern_meta` module emulates the CERN HTTPD Meta file semantics. Meta files are HTTP headers that can be output in addition to the normal range of headers for each file a user accesses.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>MetaDir</code> <i>directory</i>	Sets the name of the directory where you can find CERN-style meta information files	server, virtual, directory, <code>.htaccess</code>
<code>MetaFiles</code> On Off	Activates CERN meta-file processing	server, virtual, directory, <code>.htaccess</code>
<code>MetaSuffix</code> <i>suffix</i>	Assigns the filename suffix for the file containing CERN-style meta information	server, virtual, directory, <code>.htaccess</code>

mod_charset_lite

The `mod_charset_lite` module allows the server to change the character set of responses before sending them to the client. In an EBCDIC environment, Apache always translates HTTP protocol content (such as response headers) from the code page of the Apache process locale to ISO-8859-1, but not the body of responses. In any

environment, you can use `mod_charset_lite` to specify that response bodies should be translated. For example, if files are stored in EBCDIC, then `mod_charset_lite` can translate them to ISO-8859-1 before sending them to the client.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>CharsetDefault</code> <i>charset</i>	Sets the character-set to translate into	server, virtual, directory, <code>.htaccess</code>
<code>CharsetOptions</code> <i>option</i> [<i>option</i>] ...	Configures character-set translation behavior	server, virtual, directory, <code>.htaccess</code>
<code>CharsetSourceEnc</code> <i>charset</i>	Configures the source character-set of files	server, virtual, directory, <code>.htaccess</code>

mod_dav

The `mod_dav` module provides class 1 and class 2 WebDAV (Web-based Distributed Authoring and Versioning) functionality for Apache. This extension to the HTTP protocol

allows creating, moving, copying, and deleting resources and collections on a remote Web server.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>Dav</code> On Off <i>provider-name</i>	Enables WebDAV HTTP methods	directory
<code>DavDepthInfinity</code> On Off	Allows PROPFIND, Depth: Infinity requests	server, virtual, directory
<code>DavLockDB</code> <i>file-path</i>	Location of the DAV lock database	server, virtual
<code>DavMinTimeout</code> <i>seconds</i>	Minimum amount of time the server holds a lock on a DAV resource	server, virtual, directory

Apache Extended Modules and Directives (continued)



mod_dbd

The `mod_dbd` module manages SQL database connections using Apache Portable Runtime (APR) library. It provides database connections on request to modules requiring SQL

database functions, and takes care of managing databases with optimal efficiency and scalability for both threaded and non-threaded MPMs.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>DBDExptime seconds</code>	Sets the keepalive time for idle connections	server, virtual
<code>DBDKeep number</code>	Sets the maximum sustained number of connections	server, virtual
<code>DBDMax number</code>	Sets the maximum number of connections	server, virtual
<code>DBDMin number</code>	Sets the minimum number of connections	server, virtual
<code>DBDParams param1=value1[,param2=value2]</code>	Appends parameters for the database connection	server, virtual
<code>DBDPersist On Off</code>	Specifies whether to use persistent connections	server, virtual
<code>DBDPrepareSQL "SQL statement" label</code>	Defines an SQL prepared statement	server, virtual
<code>DBDriver name</code>	Specifies an SQL driver	server, virtual

mod_deflate

The `mod_deflate` module provides the DEFLATE output filter that allows output from your server to be compressed before being sent to the client over the network.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>DeflateBufferSize value</code>	Fragment size to be compressed at one time by <code>zlib</code>	server, virtual
<code>DeflateCompressionLevel value</code>	Specifies how much compression you apply to the output	server, virtual
<code>DeflateFilterNote [type] notename</code>	Places the compression ratio in a note for logging	server, virtual
<code>DeflateMemLevel value</code>	Specifies how much memory <code>zlib</code> should use for compression	server, virtual
<code>DeflateWindowSize value</code>	Sets the <code>zlib</code> compression window size	server, virtual

mod_disk_cache

The `mod_disk_cache` module implements a disk-based storage manager. You would use it primarily in conjunction with `mod_cache`.

Content is stored in and retrieved from the cache using URI-based keys. Content with access protection is not cached.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>CacheDirLength length</code>	Sets the number of characters in subdirectory names	server, virtual
<code>CacheDirLevels levels</code>	Sets the number of levels of subdirectories in the cache	server, virtual
<code>CacheMaxFileSize bytes</code>	Sets the maximum size (in bytes) of a document to be placed in the cache	server, virtual
<code>CacheMinFileSize bytes</code>	Sets the minimum size (in bytes) of a document to be placed in the cache	server, virtual
<code>CacheRoot directory</code>	Sets the directory root under which cache files are stored	server, virtual

mod_expires

The `mod_expires` module controls the setting of the Expires HTTP header and the max-age directive of the Cache-Control HTTP header in server responses. You can set the expiration date to be relative to either the time the source file was last modified, or to the time of the client access.

These HTTP headers are an instruction to the client about the document's validity and persistence. If cached, the document may be fetched from the cache rather than from the source until this time has passed. After that, the cache copy is considered expired and invalid, and a new copy must be obtained from the source.

DIRECTIVE	DESCRIPTION	CONTEXT
ExpiresActive On Off	Enables the generation of Expires headers	server, virtual, directory, .htaccess
ExpiresByType <i>MIME-type seconds</i>	Configures the value of the Expires header configured by MIME type	server, virtual, directory, .htaccess
ExpiresDefault <i>base [plus] number type</i>	Sets the default algorithm for calculating expiration time	server, virtual, directory, .htaccess

mod_ext_filter

The `mod_ext_filter` module presents a simple and familiar programming model for filters. With this module, a program that reads from `stdin` and writes to `stdout` (such as a Unix-style filter command) can be a filter for Apache. This filtering mechanism is much slower than using a filter that is specially written for the Apache API and runs inside of the Apache server process, but it does have the following benefits:

- The programming model is much simpler.
- You can use any programming/scripting language, provided that it allows the program to read from standard input and write to standard output.
- You can use existing programs, unmodified, as Apache filters.

Even when the performance characteristics are not suitable for production use, you can use `mod_ext_filter` as a prototype environment for filters.

DIRECTIVE	DESCRIPTION	CONTEXT
ExtFilterDefine <i>filtername parameters</i>	Defines an external filter characteristics	server
ExtFilterOptions <i>option [option] ...</i>	Configures a filter's options	directory

mod_headers

The `mod_headers` module provides directives to control and modify HTTP request and response headers. You can merge, replace, or remove headers.

DIRECTIVE	DESCRIPTION	CONTEXT
Header [<i>condition</i>] set append merge add unset echo edit <i>header [value]</i> [early env=(!) <i>variable</i>]	Configures HTTP response headers	server, virtual, directory, .htaccess
RequestHeader set append merge add unset edit <i>header [value]</i> [<i>replacement</i>] [early env=(!) <i>variable</i>]	Configures HTTP request headers	server, virtual, directory, .htaccess

continued

Apache Extended Modules and Directives (continued)

mod_ident

The `mod_ident` module queries an RFC 1413-compatible daemon on a remote host to look up the owner of a connection.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>IdentityCheck On Off</code>	Enables logging of the RFC 1413 identity of the remote user	server, virtual, directory
<code>IdentityCheckTimeout seconds</code>	Determines the timeout duration for identity requests	server, virtual, directory

mod_include

The `mod_include` module provides a filter that will process files before they are sent to the client. The processing is controlled by specially formatted SGML comments, referred to

as *elements*. These elements allow conditional text, the inclusion of other files or programs, and the setting and printing of environment variables.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>SSIEnableAccess On Off</code>	Enables the -A flag during conditional flow-control processing	directory, <code>.htaccess</code>
<code>SSIEndTag tag</code>	String that ends an include element	server, virtual
<code>SSIErrorMsg message</code>	Error message displayed when there is an SSI error	server, virtual, directory, <code>.htaccess</code>
<code>SSIETag On Off</code>	Controls whether the server generates ETags	directory, <code>.htaccess</code>
<code>SSILastModified On Off</code>	Controls whether the server generates Last-Modified headers	directory, <code>.htaccess</code>
<code>SSIStartTag tag</code>	String that starts an include element	server, virtual
<code>SSITimeFormat formatstring</code>	Configures the format in which date strings are displayed	server, virtual, directory, <code>.htaccess</code>
<code>SSIUndefinedEcho string</code>	String displayed when an unset variable is echoed	server, virtual, directory, <code>.htaccess</code>
<code>XBitHack On Off Full</code>	Parses SSI directives in files with the execute bit set	server, virtual, directory, <code>.htaccess</code>

mod_ldap

The `mod_ldap` module was created to improve the performance of Web sites relying on back-end connections to LDAP servers. In addition to the functions provided by the standard LDAP libraries, this module adds an LDAP connection pool and an LDAP shared memory cache.

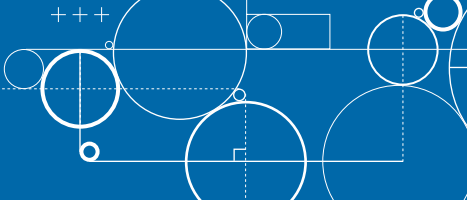
DIRECTIVE	DESCRIPTION	CONTEXT
<code>LDAPCacheEntries</code> <i>number</i>	Sets the maximum number of entries in the primary LDAP cache	server
<code>LDAPCacheTTL</code> <i>seconds</i>	Sets the time that cached items remain valid	server
<code>LDAPConnectionTimeout</code> <i>seconds</i>	Specifies the socket connection timeout in seconds	server
<code>LDAPOpCacheEntries</code> <i>number</i>	Sets the number of entries used to cache LDAP compare operations	server
<code>LDAPOpCacheTTL</code> <i>seconds</i>	Sets the time that entries in the operation cache remain valid	server
<code>LDAPSharedCacheFile</code> <i>directory/filename</i>	Sets the shared memory cache file	server
<code>LDAPSharedCacheSize</code> <i>bytes</i>	Sets the size in bytes of the shared-memory cache	server
<code>LDAPTrustedClientCert</code> <i>type directory/filename/nickname</i> [<i>password</i>]	Sets the file containing or nickname referring to a per-connection client certificate. Not all LDAP toolkits support per-connection client certificates.	server, virtual, directory, .htaccess
<code>LDAPTrustedGlobalCert</code> <i>type directory/filename</i> [<i>password</i>]	Sets the file or database containing global trusted Certificate Authority or global client certificates	server
<code>LDAPTrustedMode</code> <i>type</i>	Specifies the SSL/TLS mode to be used when connecting to an LDAP server	server, virtual
<code>LDAPVerifyServerCert</code> On Off	Forces server certificate verification	server

mod_log_forensic

The `mod_log_forensic` module provides for forensic logging of client requests. Logging is done before and after processing a request, so the forensic log contains two log lines for each request.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>ForensicLog</code> <i>filename pipe</i>	Sets the filename of the forensic log	server, virtual

Apache Extended Modules and Directives (continued)



mod_mem_cache

The `mod_mem_cache` module requires the service of `mod_cache`. It acts as a support module for `mod_cache` and provides a memory-based storage manager. You can configure `mod_mem_cache` to operate in two modes: caching open file descriptors or caching objects in heap storage. The

`mod_mem_cache` module is most useful when you use it to cache locally generated content or to cache back-end server content for `mod_proxy` configured for ProxyPass (also called reverse proxy).

DIRECTIVE	DESCRIPTION	CONTEXT
<code>MCacheMaxObjectCount</code> <i>value</i>	Sets the maximum number of objects allowed to be placed in the cache	server
<code>MCacheMaxObjectSize</code> <i>bytes</i>	Sets the maximum size (in bytes) of a document allowed in the cache	server
<code>MCacheMaxStreamingBuffer</code> <i>bytes</i>	Sets the maximum amount of a streamed response to buffer in memory before declaring the response uncacheable	server
<code>MCacheMinObjectSize</code> <i>bytes</i>	Sets the minimum size (in bytes) of a document to be allowed in the cache	server
<code>MCacheRemovalAlgorithm</code> <code>LRU GDSF</code>	Configures the algorithm used to select documents for removal from the cache	server
<code>MCacheSize</code> <i>KBytes</i>	Sets the maximum amount of memory used by the cache in KBytes	server

mod_mime_magic

The `mod_mime_magic` module determines the MIME type of files in the same way the Unix `file(1)` command works: it looks at the first few bytes of the file. It is intended as a second

line of defense for cases where `mod_mime` cannot resolve the MIME type.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>MimeMagicFile</code> <i>file-path</i>	Enables MIME-type determination based on file contents using the specified magic file	server, virtual

mod_proxy

The `mod_proxy` module implements a proxy/gateway for Apache. It implements proxying capability for AJP13 (Apache JServe Protocol version 1.3), FTP, CONNECT (for SSL),

HTTP/0.9, HTTP/1.0, and HTTP/1.1. You can configure the module to connect to other proxy modules for these and other protocols.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>AllowCONNECT port [port] ...</code>	Configures the ports that are allowed to connect through the proxy	server, virtual
<code>BalancerMember [balancerurl] url [key=value [key=value ...]]</code>	Adds a member to a load balancing group	directory
<code>NoProxy host [host] ...</code>	Configures the hosts, domains, or networks that will be connected to directly	server, virtual
<code><Proxy wildcard-URL> ...</Proxy></code>	Defines a container for directives applied to proxied resources	server, virtual
<code>ProxyBadHeader IsError Ignore StartBody</code>	Determines how to handle bad header lines in a response	server, virtual
<code>ProxyBlock * word host domain [word host domain] ...</code>	Configures the words, hosts, or domains that are banned from being proxied	server, virtual
<code>ProxyDomain domain</code>	Sets the default domain name for proxied requests	server, virtual
<code>ProxyErrorOverride On Off</code>	Overrides error pages for proxied content	server, virtual
<code>ProxyFtpDirCharset character-set</code>	Defines the character set for proxied FTP listings	server, virtual, directory
<code>ProxyIOBufferSize bytes</code>	Determines the size of the internal data throughput buffer	server, virtual
<code><ProxyMatch regex> ...</ProxyMatch></code>	Defines a container for directives applied to regular-expression-matched proxied resources	server, virtual
<code>ProxyMaxForwards number</code>	Sets the maximum number of proxies that a request can be forwarded through	server, virtual
<code>ProxyPass [path] ! URL [key=value key=value ...]] [nocanon] [interpolate]</code>	Maps remote servers into the local server URL-space	server, virtual, directory
<code>ProxyPassInterpolateEnv On Off</code>	Enables Environment Variable interpolation in Reverse Proxy configurations	server, virtual, directory
<code>ProxyPassMatch [regex] ! URL [key=value [key=value ...]]</code>	Maps remote servers into the local server URL-space using regular expressions	server, virtual, directory
<code>ProxyPassReverse [path] URL [interpolate]</code>	Adjusts the URL in HTTP response headers sent from a reverse-proxied server	server, virtual, directory
<code>ProxyPassReverseCookieDomain internal-domain public-domain [interpolate]</code>	Adjusts the Domain string in Set-Cookie headers from a reverse-proxied server	server, virtual, directory
<code>ProxyPassReverseCookiePath internal- path public-path [interpolate]</code>	Adjusts the Path string in Set-Cookie headers from a reverse-proxied server	server, virtual, directory
<code>ProxyPreserveHost On Off</code>	Uses an incoming Host HTTP request header for a proxy request	server, virtual
<code>ProxyReceiveBufferSize bytes</code>	Sets the network buffer size for proxied HTTP and FTP connections	server, virtual
<code>ProxyRemote match remote-server</code>	Configures the remote proxy used to handle certain requests	server, virtual

(continued)

continued



Apache Extended Modules and Directives (continued)

mod_proxy (continued)

DIRECTIVE	DESCRIPTION	CONTEXT
ProxyRemoteMatch <i>regex remote-server</i>	Configures the remote proxy used to handle requests matched by regular expressions	server, virtual
ProxyRequests On Off	Enables forward (standard) proxy requests	server, virtual
ProxySCGIInternalRedirect On Off	Enables internal redirect responses from the back-end	server, virtual, directory
ProxySCGISendfile On Off <i>headername</i>	Enables evaluation of an X-Sendfile pseudo response header	server, virtual, directory
ProxySet <i>URL key=value [key=value ...]</i>	Sets various Proxy balancer or member parameters	directory
ProxyStatus Off On Full	Shows Proxy LoadBalancer status in <code>mod_status</code>	server, virtual
ProxyTimeout <i>seconds</i>	Sets the network timeout for proxied requests	server, virtual
ProxyVia On Off Full Block	Shows information provided in the Via HTTP response header for proxied requests	server, virtual

mod_rewrite

The `mod_rewrite` module uses a rule-based rewriting engine (based on a regular-expression parser) to rewrite requested URLs on the fly. It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule, to provide an extremely flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests, of server variables, environment variables, HTTP headers, or timestamps. You can even use external database lookups in various formats to achieve highly granular URL matching.

This module operates on the full URLs (including the path-information part) both in per-server context (`httpd.conf`) and per-directory context (`.htaccess`) and can generate query-string modifications on the result. The rewritten result can lead to internal subprocesses, external request redirection, or even an internal proxy throughput.

You can find further details, discussion, and examples in the detailed `mod_rewrite` documentation at <http://httpd.apache.org/docs/2.2/rewrite/>.

DIRECTIVE	DESCRIPTION	CONTEXT
RewriteBase <i>URL-path</i>	Sets the base URL for per-directory rewrites	directory, <code>.htaccess</code>
RewriteCond <i>TestString CondPattern</i>	Defines a condition under which rewriting will take place	server, virtual, directory, <code>.htaccess</code>
RewriteEngine On Off	Enables or disables the runtime rewriting engine RewriteMap synchronization	server, virtual, directory, <code>.htaccess</code>
RewriteLock <i>file-path</i>	Sets the name of the lock file used for RewriteMap synchronization	server
RewriteLog <i>file-path</i>	Sets the name of the file used for logging rewrite engine processing	server, virtual
RewriteLogLevel <i>level</i>	Sets the verbosity of the log file used by the rewrite engine	server, virtual
RewriteMap <i>MapName MapType:MapSource</i>	Defines a mapping function for key-lookup	server, virtual
RewriteOptions <i>Options</i>	Sets some special options for the rewrite engine	server, virtual, directory, <code>.htaccess</code>
RewriteRule <i>Pattern Substitution [flags]</i>	Defines rules for the rewriting engine	server, virtual, directory, <code>.htaccess</code>

mod_ssl

The `mod_ssl` module provides SSL v2/v3 and TLS v1 support for the Apache HTTP Server. It was contributed by Ralf S. Engelschall based on his `mod_ssl` project and originally derived from work by Ben Laurie. This module relies on OpenSSL to provide the cryptography engine.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>SSLCACertificateFile</code> <i>file-path</i>	Sets the file of concatenated PEM-encoded CA Certificates for Client Auth	server, virtual
<code>SSLCACertificatePath</code> <i>directory-path</i>	Sets the directory of PEM-encoded CA Certificates for Client Auth	server, virtual
<code>SSLCADNRequestFile</code> <i>file-path</i>	Sets the file of concatenated PEM-encoded CA Certificates for defining acceptable CA names	server, virtual
<code>SSLCADNRequestPath</code> <i>directory-path</i>	Sets the directory of PEM-encoded CA Certificates for defining acceptable CA names	server, virtual
<code>SSLCARevocationFile</code> <i>file-path</i>	Sets the file of concatenated PEM-encoded CA CRLs for Client Auth	server, virtual
<code>SSLCARevocationPath</code> <i>directory-path</i>	Sets the directory of PEM-encoded CA CRLs for Client Auth	server, virtual
<code>SSLCertificateChainFile</code> <i>file-path</i>	Sets the file of PEM-encoded Server CA Certificates	server, virtual
<code>SSLCertificateFile</code> <i>file-path</i>	Sets the server PEM-encoded X.509 Certificate file	server, virtual
<code>SSLCertificateKeyFile</code> <i>file-path</i>	Sets the server PEM-encoded Private Key file	server, virtual
<code>SSLCipherSuite</code> <i>cipher-spec</i>	Defines the cipher Suite available for negotiation in an SSL handshake	server, virtual, directory, .htaccess
<code>SSLCryptoDevice</code> <i>engine</i>	Enables use of a cryptographic hardware accelerator	server
<code>SSLEngine</code> <code>on</code> <code>off</code> optional	Enables the SSL operation switch	server, virtual
<code>SSLHonorCipherOrder</code> <i>flag</i>	Configures the server's cipher preference order	server, virtual
<code>SSLInsecureRenegotiation</code> <code>On</code> <code>Off</code>	Enables support for insecure renegotiation	server, virtual
<code>SSLMutex</code> <i>type</i>	Configures the semaphore for internal mutual exclusion of operations	server
<code>SSLOptions</code> <code>[+ -]option ...</code>	Configures various SSL engine run-time options	server, virtual, directory, .htaccess
<code>SSLPassPhraseDialog</code> <i>type</i>	Configures the type of pass-phrase dialog for encrypted private keys	server
<code>SSLProtocol</code> <code>[+ -]protocol ...</code>	Configures usable SSL protocol flavors	server, virtual
<code>SSLProxyCACertificateFile</code> <i>file-path</i>	Sets the file of concatenated PEM-encoded CA Certificates for Remote Server Auth	server, virtual
<code>SSLProxyCACertificatePath</code> <i>directory-path</i>	Sets the directory of PEM-encoded CA Certificates for Remote Server Auth	server, virtual
<code>SSLProxyCARevocationFile</code> <i>file-path</i>	Sets the file of concatenated PEM-encoded CA CRLs for Remote Server Auth	server, virtual
<code>SSLProxyCARevocationPath</code> <i>directory-path</i>	Sets the directory of PEM-encoded CA CRLs for Remote Server Auth	server, virtual
<code>SSLProxyCheckPeerCN</code> <code>On</code> <code>Off</code>	Specifies whether to check the remote server certificate's CN field	server, virtual

(continued)

continued

Apache Extended Modules and Directives (continued)

mod_ssl (continued)

DIRECTIVE	DESCRIPTION	CONTEXT
SSLProxyCheckPeerExpire On Off	Specifies whether to check if a remote server certificate is expired	server, virtual
SSLProxyCipherSuite <i>cipher-spec</i>	Configures the cipher Suite available for negotiation in SSL proxy handshake	server, virtual, directory, .htaccess
SSLProxyEngine On Off	Enables the SSL Proxy engine operation	server, virtual
SSLProxyMachineCertificateFile <i>filename</i>	Sets the file of concatenated PEM-encoded client certificates and keys to be used by the proxy	server
SSLProxyMachineCertificatePath <i>directory</i>	Sets the directory of PEM-encoded client certificates and keys to be used by the proxy	server
SSLProxyProtocol [+ -]protocol ...	Configures usable SSL protocol flavors for proxy usage	server, virtual
SSLProxyVerify <i>level</i>	Type of remote server certificate verification	server, virtual, directory, .htaccess
SSLProxyVerifyDepth <i>number</i>	Sets the maximum depth of CA Certificates in Remote Server Certificate verification	server, virtual, directory, .htaccess
SSLRandomSeed <i>context source</i> [bytes]	Assigns a Pseudo Random Number Generator (PRNG) seeding source	server
SSLRenegBufferSize <i>bytes</i>	Sets the size for the SSL renegotiation buffer	directory, .htaccess
SSLRequire <i>expression</i>	Allows access only when an arbitrarily complex Boolean expression is true	directory, .htaccess
SSLRequireSSL	Denies access when SSL is not used for the HTTP request	directory, .htaccess
SSLSessionCache <i>type</i>	Configures the type of global/inter-process SSL Session Cache	server
SSLSessionCacheTimeout <i>seconds</i>	Sets the number of seconds before an SSL session expires in the Session Cache	server, virtual
SSLStrictSNIVHostCheck On Off	Specifies whether to allow non-SNI clients to access a name-based virtual host	server, virtual
SSLUserName <i>varnames</i>	Assigns an environment variable name to determine the “user” field in the Apache request object	server, virtual, .htaccess
SSLVerifyClient <i>level</i>	Sets the type of Client Certificate verification	server, virtual, directory, .htaccess
SSLVerifyDepth <i>number</i>	Sets the maximum depth of CA Certificates in Client Certificate verification	server, virtual, directory, .htaccess

mod_speling

Sometimes the core Apache server cannot serve requests for documents because the request was misspelled or incorrectly capitalized. The `mod_speling` module addresses this problem by trying to find a matching document. It does its work by comparing each document

name in the requested directory against the requested document name without regard to case, and allowing up to one misspelling (character insertion / omission / transposition or wrong character). A list is built with all document names that were matched using this strategy.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>CheckCaseOnly On Off</code>	Limits the action of the module to case corrections	server, virtual, directory, .htaccess
<code>CheckSpelling On Off</code>	Enables the spelling module	server, virtual, directory, .htaccess

mod_substitute

The `mod_substitute` module provides a mechanism to perform both regular-expression and fixed-string substitutions on response bodies.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>Substitute s/pattern/substitution/[flags]</code>	Sets a pattern to filter the response content	directory, .htaccess

mod_suexec

The `mod_suexec` module allows for CGI programs to run as a pre-determined user and group, with limited privileges, on the Web server.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>SuexecUserGroup User Group</code>	Assigns a user and group for CGI programs	server, virtual

Apache Extended Modules and Directives (continued)

mod_usertrack

Previous releases of Apache have included a module that generates a *clickstream* log of user activity on a site using cookies. This was called the “cookies” module, `mod_cookies`.

In Apache 1.2 and later, this module has been renamed the “user tracking” module, `mod_usertrack`. This module has been simplified and new directives have been added.

DIRECTIVE	DESCRIPTION	CONTEXT
CookieDomain <i>domain</i>	Sets the domain to which the tracking cookie applies	server, virtual, directory, .htaccess
CookieExpires <i>expiry-period</i>	Sets the expiry time for the tracking cookie	server, virtual, directory, .htaccess
CookieName <i>token</i>	Assigns the name of the tracking cookie	server, virtual, directory, .htaccess
CookieStyle Netscape Cookie Cookie2 RFC2109 RFC2965	Configures the format of the cookie header field	server, virtual, directory, .htaccess
CookieTracking On Off	Enables a tracking cookie	server, virtual, directory, .htaccess

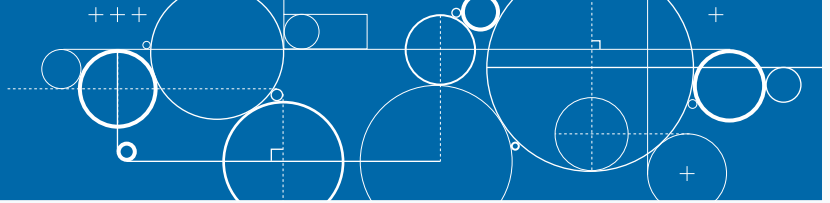
mod_vhost_alias

The `mod_vhost_alias` module creates dynamically configured virtual hosts, by allowing the IP address and/or the `Host` header of the HTTP request to be used as part of

the pathname to determine what files to serve. This allows for easy use of a huge number of virtual hosts with similar configurations.

DIRECTIVE	DESCRIPTION	CONTEXT
<code>VirtualDocumentRoot</code> <i>interpolated-directory</i> None	Configures the location of the document root for a given virtual host	server, virtual
<code>VirtualDocumentRootIP</code> <i>interpolated-directory</i> none	Configures the location of the document root for a given virtual host	server, virtual
<code><VirtualHost addr[:port] [addr[:port]] ...> ... </VirtualHost></code>	Contains directives that apply only to a specific hostname or IP address	server
<code>VirtualScriptAlias</code> <i>interpolated-directory</i> None	Configures the location of the CGI directory for a given virtual host	server, virtual
<code>VirtualScriptAliasIP</code> <i>interpolated-directory</i> None	Configures the location of the CGI directory for a given virtual host	server, virtual

Useful Perl Modules



One of the strengths of Perl is its extensive list of add-on modules that you can import into any Perl script. Perl has enjoyed several years of popularity among developers, so much so that many have contributed back to the Perl community by publishing their own custom modules onto the CPAN repository for others to learn and benefit from. Some of these modules are pre-installed on your computer alongside the Perl interpreter, while you must

manually install other modules using programs such as CPAN, ActiveState PPM, or some other packaging system. For more information on installing and using third-party modules, see Chapter 9.

You can always confirm whether you have a module installed by using `Perldoc` to view its documentation. Run the following command in a Terminal window:

```
perldoc module
```

Viewing Data

Depending on your variable types, you will find that viewing their data values is not always consistent. You can easily view scalars with `print`, but arrays require `print` and `join`, and hashes require a `while` loop with `each`. You cannot view multi-dimensional array or hash references without complex logic, unless you use some help.

Data::Dumper

The `Data::Dumper` module displays the contents of any type of Perl variable as Perl syntax. When used on a multi-dimensional array reference or a hash reference variable, each level is described in the module's output. Recursively linked references are handled in a way that eliminates infinite loops, but that accurately portrays all the information in the variable.

You can use this module with a simple, function-based interface, or with an extended, object-orientated interface. In simple mode, the output appears with generic variable names on the top level, but all keys and values are accurate. In extended mode, you can give the dumped references user-specified names:

```
use Data::Dumper;
print Dumper( $hashref, ... );
```

Because `Data::Dumper` outputs literal Perl syntax, you can use `eval` to recreate the original structure as a new variable in memory:

```
$VAR1 = {
  'key1' => value1,
  'key2' => value2,
  ...
};
```

This module is standard in all Perl installations.

FUNCTIONS	DESCRIPTION
<code>Dumper (LIST)</code>	Returns all keys, values, and references found in <i>LIST</i> as a string.
CONFIGURATION VARIABLES	DESCRIPTION
<code>\$Data::Dumper::Indent</code>	Controls the style of indentation.
<code>\$Data::Dumper::Varname</code>	Assigns the prefix to use for tagging variable names in the output. The default is "VAR."
<code>\$Data::Dumper::Useqq</code>	Enables the use of double quotes for representing string values when set.
<code>\$Data::Dumper::Terse</code>	Allows <code>Data::Dumper</code> to emit single, non-self-referencing values as atoms and terms, rather than statements.
<code>\$Data::Dumper::Deepcopy</code>	Allows deep copies of structures; cross-referencing will then only be done when absolutely essential.
<code>\$Data::Dumper::Quotekeys</code>	Enables controlling whether hash keys are quoted.
<code>\$Data::Dumper::Pair</code>	Can be set to a string that specifies the separator between hash keys and values. The default is "=>".
<code>\$Data::Dumper::Maxdepth</code>	Set to a positive integer that specifies the depth beyond which we do not venture into a structure.
<code>\$Data::Dumper::Sortkeys</code>	Set to a Boolean value to control whether hash keys are dumped in sorted order.
<code>\$Data::Dumper::Deparse</code>	Set to a Boolean value to control whether code references are turned into Perl source code.

Securing Data

You can secure your data within Perl by making a digest fingerprint of the data (ensuring reliability) or encrypting it entirely (ensuring privacy).

Digest::MD5

The Digest::MD5 module allows you to use the RSA Data Security Inc. MD5 algorithm from within Perl programs. The algorithm takes as input a string of arbitrary length and produces as output a 128-bit “fingerprint,” or *message digest*, of the input. You can use this fingerprint later to validate that the original input string has not been compromised. This module provides the same functionality as the `md5sum` command-line program on Unix systems.

The Digest::MD5 module provides a function interface and an object-orientated interface. None of these functions are exported by default, so you must specify them when importing the module:

```
use Digest::MD5 qw( md5 md5_hex md5_base64 );
$md5sum = md5_hex( $data );
```

This module is standard in all Perl installations.

FUNCTIONS	DESCRIPTION
<code>md5(data)</code>	Calculates the MD5 message digest of the data, and returns it in binary form
<code>md5_hex(data)</code>	Same as <code>md5</code> , but returns the digest in hexadecimal form
<code>md5_base64(data)</code>	Same as <code>md5</code> , but returns the digest as a base64 encoded string

Note that the MD5 algorithm is not as strong as it used to be. Since 2005, it has been easy to generate different messages that produce the same MD5 digest. Stronger digest modules include Digest::SHA1 and Digest::HMAC. Read the PerlDoc Digest page for more information on alternatives to Digest::MD5.

Crypt::CBC

The Crypt::CBC module is a Perl-only implementation of the cryptographic cipher block chaining (CBC) mode. In combination with a block cipher such as DES or Blowfish, you can encrypt and decrypt strings of arbitrary length.

To use this module, you first create a Crypt::CBC cipher object with `new`. At the time of cipher creation, you must specify a *shared-secret key* to use and, optionally, a block encryption algorithm. A shared-secret key is a string that you use to encrypt and decrypt using a cipher algorithm. Both the encrypting and decrypting programs require this key in order to create and access the original data:

```
use Crypt::CBC;
my $cipher = Crypt::CBC->new(
    -key => SECRET,
    -cipher => 'Blowfish',
);
$ciphertext = $cipher->encrypt( $data );
$plaintext = $cipher->decrypt( $ciphertext );
```

The `-key` argument provides either a passphrase to use to generate the encryption key, or the literal value of the block cipher key. If used in passphrase mode, which is the default, `-key` can be any number of characters; the actual key is derived by sending the passphrase through a series of MD5 hash operations.

The `-cipher` option specifies which block cipher algorithm to use to encode each section of the message. This argument is optional and defaults to the DES algorithm unless you specify otherwise.



Useful Perl Modules (continued)

If you only have a single variable of data that requires encryption or decryption, you can use `encrypt` or `decrypt`. However, if you have several pieces of data, this can be inefficient and slow because the CBC

algorithm constantly rebuilds itself for each method call. Instead, you can call `start` once, call `crypt` as many times as needed, and then call `finish`.

METHODS	DESCRIPTION
<code>new(args)</code>	Creates a new <code>Crypt::CBC</code> object; you can use arguments to assign the key, cipher, and other options.
<code>start(process)</code>	Prepares the cipher for a series of encryption or decryption steps, resetting the internal state of the cipher if necessary; the <code>process</code> field must literally be “encrypting” or “decrypting.”
<code>crypt(data)</code>	Encrypts or decrypts the desired data.
<code>finish()</code>	Flushes the internal buffer and returns any leftover ciphertext.
<code>encrypt(data)</code>	Runs the entire sequence of <code>start</code> , <code>crypt</code> , and <code>finish</code> for you, processing the provided plaintext and returning the corresponding ciphertext.
<code>decrypt(data)</code>	Runs the entire sequence of <code>start</code> , <code>crypt</code> , and <code>finish</code> for you, processing the provided plaintext and returning the corresponding plaintext.
<code>encrypt-hex(data)</code>	Runs <code>encrypt</code> , but returns the ciphertext in hexadecimal characters.
<code>decrypt-hex(data)</code>	Runs <code>decrypt</code> , but accepts hexadecimal ciphertext as input.

This type of encryption becomes useless if your shared-secret key becomes exposed. A more secure form of encryption is called *public-private key* cryptography. This involves encrypting data with your private key, and decrypting it with your public key. For more information, see the Perl module `Crypt::OpenSSL::RSA`.

Storing Data

Perl allows you to store and retrieve data using various methods and formats. This is useful if you need to write data to disk, and access it later from a separate Perl process.

Storable

The `Storable` package brings persistence to your Perl data structures that contain any combination of scalar, array, hash, or reference objects. It allows you to serialize the structure and save it to disk, and recreate the original structure in memory when reading from disk.

You can use it in the regular procedural way by calling `store` with a reference to the object to be stored, along with the filename where the image should be written. To retrieve data stored to disk, use `retrieve` with a filename:

```
use Storable;
store( $ref, 'filename' );
$ref = retrieve( 'filename' );
```

FUNCTIONS	DESCRIPTION
<code>store(REF, file)</code>	Serializes the reference variable into a data stream and writes it to disk as <i>file</i> .
<code>nstore(REF, file)</code>	Serializes the reference variable, but writes it to disk as big-endian data, also known as network-order.
<code>retrieve (file)</code>	Retrieves the serialized data from <i>file</i> and reconstructs the original reference variable. This function works with both <code>store</code> and <code>nstore</code> .
<code>freeze (REF)</code>	Serializes the reference variable into memory only.
<code>thaw(VAR)</code>	Retrieves serialized data from a variable created with <code>freeze</code> .

Additional file locking is available with the methods `lock_store`, `lock_nstore`, and `lock_retrieve`. These functions are not exported by default. To use them, you must provide the function name as an argument when importing `Storable` into your Perl script.

Storing Data *(continued)***Compress::ZLib**

The `Compress::ZLib` module provides a Perl interface to the `zlib` compression library. This module can be split into two general areas of functionality: a simple read/write interface to `gzip` files and a low-level in-memory compression/decompression interface:

```
use Compress::ZLib;
$zdata = compress( $data );
$zdata = uncompress( $zdata );
```

FUNCTIONS	DESCRIPTION
<code>compress</code> (<code>VAR</code> , level)	Compresses the input data with <code>zlib</code> , and sets the optional level parameter to 0 through 9 for no-compression to best-compression
<code>uncompress</code> (<code>VAR</code>)	Uncompresses data that has been compressed using <code>zlib</code>

If you are interested in compressing a stream of data using `zlib`, see the `Compress::ZLib` PerlDoc page for syntax and examples.

Manipulating Data

You can manipulate data from one format or unit into another very easily in Perl. Two examples of calculation, currencies and dates, are demonstrated here.

Finance::Currency::Convert

You can use `Finance::Currency::Convert` to retrieve live exchange rates provided by `Finance::Quote` and calculate the conversion rate of an amount from one monetary currency into another. This module uses the standardized three-character ISO currency codes as currency symbols, such as EUR for euros, USD for United States dollars, and CAD for Canadian dollars.

After you initialize the module, you can use either `updateRate` or `updateRates` with a list of symbols to retrieve the latest prices from Yahoo! Finance. `updateRate` only retrieves the exchange rates between the specified symbols; `updateRates` retrieves all rate combinations that have a quote available involving the specified symbols:

```
use Finance::Currency::Convert;
my $conv = new Finance::Currency::Convert;
$conv->updateRate( $fromSymbol, $toSymbol );
print $conv->convert( $amount, $fromSymbol, $toSymbol );
```

You can cache the retrieved rates by storing them to disk.

You do this with `setRatesFile` and `writeRatesFile`. This is helpful because `updateRates` may take some time downloading all possible values. You could set up a cache that automatically expires by comparing the time of the rates file:

```
my $file = ".rates";
my @stat = stat( $file );
$conv->setRatesFile( $file );
if ( time - $stat[ 9 ] > 600 ) {
    $conv->updateRates( $fromSymbol, $toSymbol );
    $conv->writeRatesFile();
}
print $conv->convert( $amount, $fromSymbol, $toSymbol );
```

This example code uses the `stat` function to retrieve the file-system status of the rates file. This function returns an array of 13 items, each representing a specific piece of information. In the ninth index is the file's "last modified" timestamp in seconds. By comparing that timestamp to the current time, you can force the rate to be updated; in this example, the rates are updated every 600 seconds, or ten minutes.

Useful Perl Modules (continued)

Manipulating Data *(continued)*

Date::Manip

You can use the `Date::Manip` module to read a date string from virtually any format, compose a date string into any format, or even convert a date between different languages. This is a robust date calculator that allows you to calculate the difference in time between two dates, or calculate a future date represented as an offset of time. `Date::Manip` even understands exotic and obscure date queries referenced only by a proper name, like local statutory and religious holidays, when you use it to specify an absolute date or range.

For example, you can create a program that counts number of business days — non-holiday weekdays — between today and three months from now. `Date::Manip` will give you an appropriately calculated result regardless of the day of the week, or the time of the year, you run the program:

```
use Date::Manip;
my $date = ParseDate( "string" );
print UnixDate( $date, "format" );
```

In this example, `Date::Manip` provides `ParseDate`, which you can use to parse a complex date string such as “2pm,

2nd Tuesday in June 2010” or “06/08/10 14:00:00” and store it into a special `$date` variable. You can then display that date using `UnixDate` with various formatting macros and literal characters; for example, “%a, %E of %B %Y @ %H:%M” produces “Tue, 8th of June 2010 @ 14:00.” For the complete list of format macros, see the `Date::Manip` PerlDoc page.

Deltas represent an arbitrary amount of time, not a specific date. You can use `ParseDateDelta` to create a `$delta` variable that you can use in a calculation:

```
my $delta = ParseDateDelta( "string" );
my $date2 = DateCalc( $date, $delta );
print UnixDate( $date2, "format" );
```

Calculations with `DateCalc` can combine a date and a delta to produce a second date, or two dates to produce a delta. For example, you could parse delta string “+45 hours 15 minutes” and combine it with the previous example date. `Date::Manip` is smart enough to identify values based in one unit and automatically convert them to larger units. Using the same `UnixDate` format as before, your new date would display as “Thu, 10th of June 2010 @ 11:15.”

Perl Libraries

Many libraries have been written for Perl that you can leverage in your Perl scripts. A library is a collection of modules that, when used together, provides an easy way to execute complex functionality in Perl.

CGI

The Common Gateway Interface (CGI) library uses Perl objects to make it easy to create Web fill-out forms and parse their contents. This library allows you to create a CGI reference object handle, an entity that contains the values of the current query string and other CGI state variables. Using a CGI object's methods, you can examine keywords and parameters passed to your script, and create forms whose initial values are taken from the current query (thereby preserving state information). The module provides shortcut functions that produce boilerplate HTML, thus reducing typing and coding errors. It also provides functionality for some of the more advanced features of CGI scripting, including support for file uploads, cookies, cascading style sheets, server push, and frames.

This library is described extensively in Chapter 12, but you can always read the “CGI” PerlDoc page for more information.

MODULE	DESCRIPTION
CGI	Handles Common Gateway Interface requests and responses
CGI::Carp	Provides routines for writing to the Apache error log
CGI::Cookie	Provides an interface to Web browser cookies
CGI::Pretty	Produces nicely formatted HTML code
CGI::Push	Provides a simple interface to Server Push

On CPAN, there are hundreds of additional modules written by third-party developers that further enhance the CGI library, providing specialized functionality for your CGI Perl scripts. Go to search.cpan.org and search for “CGI” to bring up the complete list.

Perl Libraries (continued)

DBI

The Database Independent (DBI) library is an access module for the Perl programming language. It defines a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used.

It is important to remember that the DBI is just an interface; it is a layer of “glue” between an application and one or more database driver modules. It is the driver modules that do most of the real work. The DBI library provides a standard interface and framework for the drivers to operate within.

MODULE	DESCRIPTION
DBI	Provides the main database-independent interface for Perl
DBI::DBD	Reads the Perl DBI database driver writer’s guide
DBI::FAQ	Reads the frequently asked questions for the Perl5 Database Interface
DBI::Profile	Applies performance profiling and benchmarking for the DBI
DBI::ProxyServer	Configures a server for the DBD::Proxy driver
DBD::DBM	Installs a DBI driver for DBM and multi-level DBM database files
DBD::File	Importable base class used for writing DBI drivers
DBD::Gofer	Configures a stateless-proxy driver for communicating with a remote DBI
DBD::Proxy	Configures a proxy driver for the DBI
DBD::Sponge	Creates a DBI statement handle from Perl data

You will require a DBD module specific to your choice of database server software. For an example of implementing DBI in your Perl scripts, Chapter 21 demonstrates how to use DBI and DBD::mysql to connect to a MySQL database server. For a complete list of DBD modules, go to <http://search.cpan.org> and search for “DBD” to see what is available.

LWP

The libwww-perl (LWP) collection is a set of Perl modules that provides a simple and consistent programming interface to the World Wide Web. This library contains modules and functions that allow you to create HTTP clients to query Web sites, just like a normal Web browser. You can even implement a simple HTTP daemon, just like a normal Web server.

MODULE	DESCRIPTION
LWP	Provides the main LWP library interface
LWP::Simple	Provides a simplified procedural interface to LWP
LWP::UserAgent	Configures the Web user agent class
Net::HTTP	Implements Low-level HTTP connection (client)
Net::HTTPS	Implements Low-level HTTPS connection (client)
File::Listing	Parses a file-system directory listing
HTTP::Config	Interacts with for HTTP request and response objects
HTTP::Cookies	Implements HTTP cookie jars
HTTP::Daemon	Produces a simple HTTP server
HTTP::Headers	Interfaces with encapsulating HTTP message headers
HTTP::Message	Interfaces with HTTP style message
HTTP::Negotiate	Interfaces with HTTP content negotiation algorithm
HTTP::Request	Interfaces with HTTP style request message
HTTP::Response	Interfaces with HTTP style response message
HTTP::Status	Interfaces with HTTP status code processing

INDEX

SYMBOLS

+ (add operator), 78
&& (AND command), 55
<> (angle brackets), 54
-> (arrow operator), 54
\ (backslash), 55
> (greater-than), 55
>= (greater-than or equal-to), 55
< (less than), 55
<= (less-than or equal-to), 55
{ } (curly brackets), 54
/ (divide operator), 78
== (equal-to), 55
** (exponentiation operator), 78
++,-- (increment, decrement operators), 78
&& (logical operator), 79
// (logical operator), 79
|| (logical operator), 79
% (modulus operator), 78
+ (multiply operator), 78
! (NOT command), 55
!= (not-equal-to), 55
|| (OR command), 55
() (parentheses), 54
'/' (quotes), 56
; (semicolons), 54
[] (square brackets), 54
- (subtract operator), 78

A

Active Server Pages (ASP), 14
ActivePerl, 24, 26–29, 36–39
ActivePerl Enterprise, 24
ActivePerl Perl Package Manager. *See* Perl Package Manager (PPM)
ActivePerl Pro Studio, 27
ActiveState Community License, 24
ActiveState Programmer Network, 24
Activity Feed plugin (Facebook), 239
activity logs, 138, 350, 351
add operator (+), 78
Advanced Package Tool (APT), 34
AJAX (Asynchronous JavaScript, and XML)
 calling JavaScript through Perl subroutines, 292–293
 calling Perl subroutines through JavaScript, 290–291
 CGI:::Ajax, 286–289, 294–295
 overview, 284–285

AND command (&&), 55
angle brackets (<>), 54
@Anywhere APIs, 249, 266–267
Apache. *See also specific topics*
 activity logs, 138, 350, 351
 authentication/authorization modules and directives, 398–403
 base modules and directives, 391–397
 CGI Handler, 12, 13, 134–137
 comparing to other Web servers, 16–17
 configuration directives, 128, 337, 386–390
 configuring, 44–45, 50–51, 342–345
 creating user directories, 130–133
 directives, 386–389
 documentation, 23
 downloading for Windows, 40–41
 extended modules and directives, 404–417
 history of, 2
 installing, 42–43, 48, 49
 limiting CGI access in, 348–349
 restarting, 128, 363
 run-time configuration directives, 386–390
 service, 46–47, 52–53
 TLS/SSL on, 337
 user authentication, 208–215
 versions of, 2
 Web site, 23
Apache CGI Handler, 12, 13, 128–129
Apache CGI module, 134–135
Apache Configuration GUI for Windows, 51
Apache mod_perl module, 356–363
Apache Server Monitor, 47
Apache Service Control Panel, 47
Apache Software Foundation (Web site), 40
Apache SSI Module, 192–193
APIs (application programming interfaces), 245, 248–249, 266–267
App:::Tweet, 251
APT (Advanced Package Tool), 34
array functions, 370
array references, 92–93
arrays
 building, 96–97
 data, 68–69, 421–422
 looping, 82
 Perl, 68–71
 retrieving data from, 69, 71
 storing data into, 68, 69, 70
Arrow operator (->), 54
ASP (Active Server Pages), 14

ASP.NET, 14

assignment operators, 78, 381–382

Asynchronous JavaScript, and XML. *See* AJAX (Asynchronous JavaScript, and XML)

Authen::Captcha, 278–279

authentication. *See also* user authentication

browser-based, 208–209

defined, 398

Facebook application CGI, 245

methods, 209

modules and directives, 398–402

MySQL, 316

password files, 212–213

provider, 209

types of, 208–209

authorization, 398–403

authorized users, requiring only, 214–215

auto-linking feature, 249

auto-rotating Apache logs, 138

B

backslash character (\), 55

binary packages, 115

binary test operators, 382–383

bitwise operators, 79

brackets, 54

browser-based authentication, 208–209

browsers. *See* Web browsers

built-in functions, 56, 57, 69, 368–375

built-in Perl documentation, 25

Business::PayPal::IPN module, 299

Business::PayPal::NVP module, 299, 304–305

C

C, C++, C# languages, 15

Callback URL, 255

Canvas Callback URL, 244

Canvas Page URL, 244

capitalization, 54

Captcha tests, 278–279

carriage return character, 55

certificate authority, 336, 337, 341

CGI (Common Gateway Interface)

Apache, 12, 13, 128–129, 134–135

from CGI program's point of view, 12–13

from end-user's point of view, 6–7

executing programs, 10–11

forwarding data to programs, 11

help with developing programs, 22–23

layering with Perl, SSI, and HTML::Template, 191

module, 128, 422

receiving data from programs, 11

routines, 154

scripts, 196, 216, 232–233, 288–289, 357

source code, 6

from Web browser's point of view, 8–9

from Web server's point of view, 10–11

CGI Handler (Apache), 12, 13, 128, 134–137

CGI library

importing, 156–157

module, 422

overview, 154–155, 167

reading HTTP GET/POST parameters with, 158–159

retrieving HTTP cookies with, 162–163

storing HTTP cookies with, 160–161

CGI::Ajax

adding into Perl CGI scripts, 288–289

calling

JavaScript through Perl subroutines, 292–293

Perl subroutines through JavaScript, 290–291

enabling Debug mode in, 294–295

overview, 286–287

CGI::Carp, 164–165

character classes, 384

characters (special), 55

client-side control, 143

cloud hosting, 20

code, 18, 93

command-line, 108, 114, 317, 367

commands. *See specific commands*

commenting code, 18

comments, 55

Comments plugin (Facebook), 239

Common Gateway Interface. *See* CGI (Common Gateway Interface)

community resources, 24

compound data structures, 94–95

Comprehensive Perl Archive Network. *See* CPAN (Comprehensive Perl Archive Network)

Compress::ZLib module, 421

conditional expressions, 202–203

conditional operators, 383

conditional tests, 76, 77, 197

configuration (Apache), 128

configuration context, 386

configuration directives, 337, 386–390

Configuration Section Containers, 44, 50–51

INDEX

configuration variables, 418
content-type HTTP response headers, 4, 8–9
`continue` command, 83
cookie routine, 155
cookies (HTTP)
 overview, 8, 146–147
 retrieving, 150–151, 162–163
 storing, 148–149, 160–161
CPAN (Comprehensive Perl Archive Network)
 configuring, 110
 installing Perl modules with, 112–113
 overview, 108–109
 repository, 25
 searching for Perl modules with, 111
credit card payments, processing with PayPal, 306–307
`Crypt::CBC` module, 419
CSR, 340–341
curly brackets (`{}`), 54
Cygwin environment, 17

D

data

arrays, 68–69, 421–422
compound structures, 94–95
cookie, 146
deleting from tables, 319
displaying with `TMPL_VAR`, 176–177
form, 351
forwarding to CGI program, 11
inserting into tables, 319
reading, 12
reading from MySQL, 327
receiving
 from browsers, 10
 from CGI programs, 11
 from Web browsers, 10
 from Web servers, 9
referenced, 65
retrieving
 from Perl arrays, 69, 71
 from Perl hashes, 73, 75
 from Perl scalars, 65, 67
returned by server, 7
reviewing, 351
securing, 419
selecting from tables, 319
sending, 9, 11, 13
SQL, 330–335

storing
 overview, 420–421
 in Perl arrays, 68, 69, 70
 in Perl hashes, 72, 73, 74
 in Perl scalars, 64, 66
updating in tables, 319
user-specific, 146
user-submitted, 7, 9
viewing, 418
writing to MySQL, 327
Database Driver (DBD), 323
Database Independent library. *See* Perl DBI library
`Data::Dumper` module, 94, 101, 418
`Date::Manip` module, 422
DBD (Database Driver), 323
DBI library. *See* Perl DBI library
DBM format, 215
Debian/Ubuntu Linux
 Apache configuration directives, 337
 installing
 Apache for, 48
 MySQL for, 324
 Perl for, 34
 Perl modules in, 121
 searching for Perl modules in, 120
Debian/Ubuntu operating system, 138
Debug mode, enabling in `CGI::Ajax`, 294–295
decrement operator (`--`), 78
dedicated hosting, 21
development environment, 18, 19
digest authentication type, 208, 211
`Digest::MD5` module, 419
`Digest::SHA` model, 229
directives
 Apache authentication and authorization, 398–403
 Apache base modules and, 391–397
 Apache extended modules and, 404–417
 configuration, 337, 386–390
 core-Apache, 386–389
 Multi-Process Module (MPM), 390
 user directories, 130–133
directories, 136–137, 194–195
directory paths, securing with Apache, 210–211
disks, saving images to, 275
divide operator (`/`), 78
DNS (domain name service), 20
documentation
 ActivePerl Perl Package Manager, 114
 ActiveState, 24
 Apache, 23

- HTML::Template module, 169
- MySQL, 317
- Perl, 22–23, 25, 364–366
- Perl DBI library, 326
- Strawberry Perl, 25
- domain name service (DNS), 20
- domain registrar, 20
- dynamic content, 155, 186–187

E

- elements (SSI), 191, 196–197
- else/elseif, 80–81
- e-mail messages, 152–153
- Encrypted Web Payments (EWP), 298
- encryption. *See* TLS/SSL encryption
- encryption software, 40–41
- end tags, 171
- end users, 3, 6–7, 11
- environment variables, 5, 8, 197
- equality operators, 78, 382–383
- equal-to (==), 55
- error detection, 169
- error handling, 327
- error logs, 138, 350
- error messages, 164–165, 197
- EWP (Encrypted Web Payments), 298
- exponentiation operator (**), 78

F

- Facebook
 - adding Facebook Social plugins to Web sites, 238–239
 - creating applications with Perl, 246–247
 - enabling Facebook Connect on Web sites, 240–243
 - interaction, 245
 - plugins, 238–239
 - registering Web sites as, 236–237
- Facebook Canvas API, 244–245, 247
- Facebook Graph API, 240–243
- Facebook Markup Language (FBML), 245
- Facebook Query Language (FQL), 245
- fail2ban script, 350
- files
 - accepting for upload, 270–271
 - authentication password, 212–213
 - changes in Perl, 363
 - external, 196
 - functions, 373

- HTML::Template module, 170–171
 - importing with SSI, 198
 - module, 102–103
 - shared subroutine, 87
 - SSI-enabled HTML, 190
 - static database, 209
 - static password, 209
 - statistics, 197, 204–205
 - template, 168, 172–173
- filter (output), 192–193
- Finance::Currency::Convert module, 421
- Firebug, 295
- flow control keywords, 374
- Follow Me button, 249
- followers (Twitter), 263
- footers, 184, 186–187, 206–207
- foreach loop, 82, 84–85
- forms (HTML)
 - building, 7
 - creating, 142–143
 - encoding value of, 9
 - monitoring data, 351
 - overview, 156
- forums, 23
- FQL (Facebook Query Language), 245
- functions
 - array, 370
 - built-in, 56–57, 69, 368–375
 - Compress::ZLib module, 421
 - Data::Dumper module, 418
 - Digest::MD5 module, 419
 - file, 373
 - importing
 - CGI library routines as, 157
 - methods as, 154
 - manually accessing array data with, 69
 - scalar, 369
 - of scalar references in Perl, 92
 - Storable package, 420
 - support for loops, 83

G

- galleries (image), 280–283
- GET parameters, 144–145, 158–159
- Global URI, 128
- Gnome graphical interface, 35
- GNU General Public License (GPL), 25
- Google Chrome, 295

INDEX

`goto` command, 83
Graph API, 245
Graphical User Interface (GUI), 114, 317
greater-than (>), 55
greater-than or equal-to (>=), 55
greedy quantifiers, 385
GUI (Graphical User Interface), 114, 317

H

hash functions, 371
hash references, 93, 96–97
hashes (Perl), 72–75
header routine, 155
headers
 HTML, 4
 HTML::Template, 184, 186–187
 linking with static HTML content, 206–207
 Perl script, 57
high-level programming language, 3
hosting providers, 20–21
Hovercard, 249
How to Ask Questions the SmartWay (Raymond), 23
.htaccess file, 129
HTML (HyperText Markup Language)
 benefits of separating from Perl, 166–167
 displaying, 169
 forms, 5, 7, 142–143
 interface, 6
 login prompt, 217
 routines, 155
 source code, 6
HTML::Template module
 creating
 headers and footers, 184
 new template files, 172–173
 toolbars, 185
 displaying SQL data through, 332–333
 extending to non-HTML formats, 188–189
 importing, 174–175
 layering with CGI, SSI, and Perl, 191
 linking headers, toolbars, and footers with dynamic Perl
 content, 186–187
 overview, 167–169
 structure of files in, 170–171
HTTP. *See also* cookies (HTTP)
 GET/POST parameters, 144–145, 158–159
 headers, 4, 8–9
 server capabilities, 2

HTTPS redirectors, 337
HyperText Markup Language. *See* HTML (HyperText Markup Language)

I

IDE (Komodo Integrated Development Environment), 27
if, 80–81
IIS (Internet Information Server), 16
Image::Magick, 272–274
images
 dynamic, 276–277
 implementing Captcha tests, 278–279
 JPEG, 233
 manipulating with Image::Magick, 274
 opening with Image::Magick, 272
 producing image galleries, 280–283
 resizing/cropping with Image::Magick, 273
 saving to disk, 275
@INC array, 102
increment operator (++), 78
input fields, 155
input/output functions, 371–372
Instant Payment Notification (IPN) API, 299
Internet Developer Toolbar, 295
Internet Explorer (Microsoft), 295
Internet Information Server (IIS), 16
IP address, 21
IPN (Instant Payment Notification) API, 299

J

JavaScript. *See also* AJAX (Asynchronous JavaScript, and XML)
 @Anywhere JavaScript API, 266–267
 calling
 Perl subroutines through, 290–291
 through Perl subroutines, 292–293
 SDK, 236–237
JPEG images, 233
JSON language, 284, 297

K

keys, 72–73
“key - value” statement, 73
Komodo Integrated Development Environment (IDE), 27

L

- `last` command, 83
- less than (<), 55
- less-than or equal-to (<=), 55
- library. *See* CGI library; Perl DBI library
- libwww-perl (LWP) collection, 423
- licenses, 24, 25
- lighttpd, 17
- line feed character, 55
- line spacing/formatting, 54
- Linux
 - configuring Apache on, 50–51
 - creating user directories for Apache in, 132–133
 - downloading ActivePerl for, 36–37
 - installing
 - ActivePerl for, 38–39
 - Apache `mod_perl` module for, 359
 - starting and stopping Apache service on, 52–53
- list functions, 371
- list operators, 383
- Live Stream plugin (Facebook), 239
- log directory, 138
- logic operators, 55, 79
- login prompts, 226–227
- logs (Apache), 138–141
- loop variable, 82
- looping conditions, 77
- loops, 82–83

M

- Mac OS, 55
- macros, 205
- Mail Transport Agent (MTA), 152–153
- mathematic operators, 78, 381
- messages (e-mail), 152–153
- meta-character, 384
- methods
 - authentication, 209
 - CGI, 154
 - GET, 5
 - HTML routines, 155
 - importing as functions, 154
 - Perl, 420
 - POST, 5
 - render, 244
- Microsoft Internet Explorer, 295
- MIME (Multipurpose Internet Mail Extension) types, 8–9
- `mod_actions` module, 391

- `mod_alias` module, 391
- `mod_auth_basic` module, 398
- `mod_auth_digest` module, 398
- `mod_authn_alias` module, 399
- `mod_authn_anon` module, 399
- `mod_authn_dbd` module, 399
- `mod_authn_dbm` module, 400
- `mod_authn_default` module, 400
- `mod_authn_file` module, 400
- `mod_authnz_ldap` module, 401
- `mod_authz_dbm` module, 402
- `mod_authz_default` module, 402
- `mod_authz_groupfile` module, 402
- `mod_authz_host` module, 403
- `mod_authz_owner` module, 403
- `mod_authz_user` module, 403
- `mod_autoindex` module, 392
- `mod_cache` module, 404
- `mod_cern_meta` module, 405
- `mod_cgi` module, 393
- `mod_charset_lite` module, 405
- `mod_dav` module, 405
- `mod_dbd` module, 406
- `mod_deflate` module, 406
- `mod_dir` module, 393
- `mod_disk_cache` module, 406
- `mod_env` module, 393
- `mod_expires` module, 407
- `mod_ext_filter` module, 407
- `mod_filter` module, 394
- `mod_headers` module, 407
- `mod_ident` module, 408
- modifier flags, 385
- `mod_imagemap` module, 394
- `mod_include` module, 408
- `mod_ldap` module, 409
- `mod_log_config` module, 394
- `mod_log_forensic` module, 409
- `mod_mem_cache` module, 410
- `mod_mime` module, 395
- `mod_mime_magic` module, 410
- `mod_negotiation` module, 396
- `mod_perl` module, 356–363
- `mod_proxy` module, 411–412
- `mod_rewrite` module, 412
- `mod_setenvif` module, 396
- `mod_speling` module, 415
- `mod_ssl` module, 413–414
- `mod_status` module, 397
- `mod_substitute` module, 415

INDEX

`mod_suexec` module, 415
module reference, 106
modules. *See also* Perl modules; *specific modules*
 authentication and authorization, 398–403
 CGI library, 128, 422
 compiling, 109
 DBI (Database Independent) library, 423
 downloading, 109
 files, 102–103
 installing, 109
 keywords, 375
 libwww-perl (LWP) collection, 423
 Perl, 418–423
 searching for, 108
 testing, 109
 uninstalling, 109
 upgrading, 109
modulus operator (%), 78
`mod_userdir` module, 397
`mod_usertrack` module, 416
`mod_vhost_alias` module, 417
Mozilla Firefox, 295
MPM (Multi-Process Module directives, 390
MTA (Mail Transport Agent), 152–153
multi-dimensional references, 95, 100–101
multiply operator (*), 78
Multi-Process Module (MPM) directives, 390
Multipurpose Internet Mail Extension (MIME) types, 8–9
MySQL
 authentication, 316
 connecting to, 327, 328–329
 disconnecting from, 327
 downloading MySQL for Windows, 320–321
 installing, 322–325
 overview, 221, 316–317
 server, 326–327
 SQL syntax, 318–319
MySQL Cluster, 321
MySQL Connectors, 321
MySQL Workbench, 321
MyTwitter Perl module, 258–259

N

Name-Value Pairs (NVP) interface, 298
nesting conditions, 77
nesting references, 94
.NET, 15
`Net::Twitter` module, 250–251, 258–259
`Net::Twitter::Lite`, 251

`next` command, 83
nginx, 17
non-greedy quantifiers, 385
non-HTML formats, 188–189
NOT command (!), 55
not-equal-to (!=), 55
numeric assignment operators, 381
numeric functions, 370
NVP (Name-Value Pairs) interface, 298



OAuth protocol, 248, 254–257
objects, 156
one-liner program, 367
online resources. *See* Web sites
OpenLDAP SDK, 401
OpenSSL Library, 41
operatives (relational), 55
operator precedence, 78
operators. *See specific types*
options (command-line), 367
OR command (||), 55
output, printing to screen, 60–61
output filter, 192–193
Overview page, 364



package areas, 115
`param` routine, 155
parameters, 57, 86, 144–145, 171
parentheses (()), 54
PayPal
 creating buyer and seller sandbox accounts, 302
 overview, 298–299
 PayPal Express Checkout API, 308–311
 processing credit card payments with, 306–307
 refunding transactions, 314–315
 retrieving seller's sandbox API credentials, 303
 searching transaction history, 312
 signing up for sandbox accounts, 300–301
 using `Business::PayPal::NVP` to connect to, 304–305
 viewing transaction details, 313
PayPal API, 298
PayPal Express Checkout API, 308–311
Perl. *See also* `HTML::Template` module; *specific topics*
 activity, 141
 arrays, 68–71. *See also* arrays

- benefits of separating HTML from, 166–167
- built-in functions, 56, 57, 69, 368–375
- code, 362
- comparing to other CGI languages, 14–15
- creating Facebook applications in, 246–247
- documentation, 22–23, 25, 364–366
- executing on command-line, 367
- files, 129, 363
- history of, 3
- implementations, 167
- installing, 34–35
- integrating
 - with JSON, 297
 - with XML, 296
- layering with SSI, CGI, and `HTML::Template`, 191
- loops, 82–83
- modules, 418–423
- operations, 78–79
- operators, 380–383
- output, 129
- pre-defined variables, 376–379
- references, 92–93
- regular expressions, 384–385
- script
 - anatomy of, 57
 - creating, 58–59
 - executing, 62–63
- syntax, 54–56, 169
- user authentication
 - accessing user databases, 220–222
 - authorizing user sessions, 230–231
 - checking for session authorization, 224–225
 - creating modules, 218–219
 - displaying login prompts, 226–227
 - overview, 216–217
 - restricting access to CGI scripts, 232–233
 - storing user credentials in user database, 222–223
 - terminating sessions, 234–235
 - validating user credentials, 228–229
- using for Apache CGI, 129
- using to connect to Facebook Canvas API, 244
- versions of, 3

Perl Artistic License, 25

Perl Authentication module, 218–219

Perl Authorization module, 216–217

Perl CGI

- development, 346–347
- relationship with template files, 168
- sanitizing user content in, 352–353

- scripts
 - adding `CGI::Ajax` into, 288–289
 - securing, 216
 - speeding up, 357
 - support, 285
 - validating user content in, 354–355

Perl Database Interface (DBI), 323

Perl DBI library

- changing SQL data using, 334–335
- connecting to MySQL databases with, 328–329
- displaying SQL data through `HTML::Template`, 332–333
- overview, 317, 326–327, 422–423
- retrieving SQL data using, 330–331

Perl hashes, 72–75

Perl Interpreter, 12, 129, 137

Perl modules

- calling subroutines as methods, 106–107
- creating, 104–105, 126–127
- downloading manually, 124–125
- hierarchical structure of, 105
- installing
 - with ActivePerl Perl Package Manager, 119
 - with CPAN, 112–113
 - in Debian/Ubuntu Linux, 121
 - manually, 126–127
 - in Red Hat Linux, 123
- overview, 87, 102–103

PayPal, 299

preloading, 357

searching

- with ActivePerl Perl Package Manager for, 115, 118
- with CPAN for, 111
- in Debian/Ubuntu Linux for, 120
- manually for, 124–125
- in Red Hat Linux for, 122

Perl Package Manager (PPM)

- configuring, 116–117
- installing Perl modules with, 119
- overview, 24, 114–115
- searching for Perl modules with, 115, 118

Perl scalars, 64–67

Perl subroutines

- calling
 - JavaScript through, 106–107, 292–293
 - through JavaScript, 290–291
- manipulating variables in, 90–91
- organizing program code with, 88–89
- overview, 86–87

Perl Twitter modules, 250–251

Perl-based authentication, 216–217



INDEX

perlcc program, 63
PerlDoc, 25, 364
perldoc program, 25
PHP, comparing Perl to, 14
POST method, 5
POST parameters, 144–145, 158–159
PPM. *See* Perl Package Manager (PPM)
pre-defined variables, 56, 376–379
private key, 336, 338
production environment, 19
profiler, 295
program code, 88–89
program flow, 80–81, 84–85
programs, executing, 196, 199
progress functions, 375
project deployment, 19
protocols, 248, 254–257, 336

Q

quantifiers, 385
queues, 115
quote operators, 380
quote-lite operators, 380
quotes ('/"/), 56

R

Raymond, Eric S. (author)
How to Ask Questions the SmartWay, 23
realm, 211
real-time activity, 268–269
Red Hat Linux
 Apache configuration directives, 337
 Apache log directory, 138
 installing
 Apache for, 49
 MySQL for, 325
 Perl for, 35
 Perl modules in, 123
 searching for Perl modules in, 122
Reference page, 365–366
referenced arrays, 68
referenced data, 65
referenced hash, 72
references, 92–93. *See also specific types*
regular expression functions, 369
regular expression operators, 79
regular expression quote-lite operators, 380

regular expressions, 384–385
relational operators, 55, 78, 382
render methods, 244
repositories, 115
request headers (HTTP), 4
resources. *See* Web sites
response headers (HTTP), 4
routines, 154–155, 157. *See also specific routines*
Ruby on Rails, 15
run-time configuration directives (Apache), 386–390

S

Safari Books Online, 27
sandbox accounts, 299–302
Sandbox API credentials, 303
scalar functions, 369
scalar references, 92
scalars (Perl), 64–67
ScanErrLog script, 350
screen, printing output to, 60–61
scripts
 anatomy of, 57
 content of, 57
 defined, 3
 fail2ban, 350
 Perl CGI, 288–289
Search API, 249
secure sockets layer. *See* TLS/SSL encryption
security, 346–347. *See also* TLS/SSL encryption
semicolons (;), 54
servers. *See* Web servers
Server-Side Includes. *See* SSI (Server-Side Includes)
Service Control Panel (Apache), 47
session ID cookie, 217
sessions, 147, 217, 224–225, 230–231, 234–235
shared subroutine files, 87
shift assignment operators, 382
shutting down, 13
simple domain hosting, 20
SOAP (Simple Object Access Protocol) interface, 298
software (encryption), 40–41
special characters, 55
SQL data, 330–335
SQL database, 209
square brackets ([]), 54
SSI (Server-Side Includes)
 configuring directories to use, 194–195
 displaying file statistics with, 204–205

- elements, 196–197
- enabling
 - Apache SSI module, 192–193
 - output filter, 192–193
- executing programs with, 199
- importing files with, 198
- layering with Perl, CGI, and HTML::Template, 191
- overview, 190–191
- retrieving variables with, 201
- setting variables within, 200
- using conditional expressions with, 202–203
- SSI-enabled HTML file, 190
- SSI-enabled Server directory, 190
- SSL Certificate Signing Request, 339
- statement modifiers, 76
- static database files, 209
- static HTML content, 206–207
- static IP address, 21
- static password files, 209
- statistics (file), 197, 204–205
- status updates (Twitter), 260
- Storable module, 221
- Storable package, 420
- Strawberry Perl for Windows, 25, 30–33
- Streaming API, 249
- string assignment operators, 381
- string operators, 79
- Submit buttons (forms), 7
- subquery statements, 319
- subroutines. *See* Perl subroutines
- subtract operator (-), 78
- support. *See* documentation
- syntax
 - conditional tests, 76, 77
 - foreach loop, 82
 - Perl, 54–56, 169
 - SQL, 318–319
 - subroutines, 86–87
 - synonyms in variation, 94–95
 - template, 170, 171

T

- tab character, 55
- tables, 155, 318–319
- tag names, 171
- Template library, 167
- templates
 - caching, 169
 - controlling content, 176–177

- files, 168, 172–173
- nesting with `TMPL_INCLUDE`, 182–183
- repeating content with `TMPL_LOOP`, 180–181
- syntax, 170, 171
- test SSL certificate, 340
- testing environment, 19
- text formatting, 156
- third-generation programming language, 3
- third-party modules, 24, 25
- third-party PPM repository, 358
- third-party utilities, 51
- time function, 225, 375
- timelines (Twitter), 261
- TLS/SSL encryption
 - configuring Apache to use, 342–345
 - creating private SSL keys, 338
 - generating SSL certificate signing requests, 339
 - overview, 336–337
 - signing a CSR to create Test SSL Certificates, 340
 - submitting CSR to be signed by certificate authority, 341
- `TMPL_ELSE` element, 170, 178–179
- `TMPL_IF` element, 170, 178–179
- `TMPL_INCLUDE` element, 171, 182–183
- `TMPL_LOOP` element, 171, 180–181
- `TMPL_UNLESS` element, 170
- `TMPL_VAR` element, 170, 176–177
- toolbars, 185–187, 206–207
- transactions (PayPal), 312–315
- transport layer security. *See* TLS/SSL encryption
- Tutorial page, 364–365
- TweetBox, 249
- Twitter
 - authenticating using OAuth, 254–257
 - creating MyTwitter Perl modules that inherit `Net::Twitter`, 258–259
 - following real-time activity with Twitter Streaming API, 268–269
 - modules, 250–251
 - posting status updates, 260
 - registering applications, 252–253
 - retrieving
 - list of followers, 263
 - lists of users you follow, 262
 - timelines, 261
 - searching for content using Twitter Search API, 264–265
 - Twitter @Anywhere JavaScript API, 266–267
- Twitter API, 248–249
- Twitter Search API, 264–265
- Twitter Streaming API, 268–269



INDEX

U

Ubuntu Linux. *See* Debian/Ubuntu Linux

Unix

- carriage return character, 55
- downloading ActivePerl for, 36–37
- installing ActivePerl for, 38–39
- line feed character, 55

user authentication

- Apache
 - authentication password files, 212–213
 - overview, 208–209
 - requiring only authorized users, 214–215
 - securing directory paths with Apache, 210–211

Perl

- accessing user databases, 220–222
- authorizing user sessions, 230–231
- checking for session authorization, 224–225
- creating modules, 218–219
- displaying login prompts, 226–227
- overview, 216–217
- restricting access to CGI scripts, 232–233
- storing user credentials in user database, 222–223
- terminating sessions, 234–235
- validating user credentials, 228–229

user content, 352–355

user credentials, 217, 222–223, 228–229

user databases, 220–223

user directories, creating for Apache, 130–133

user sessions, 230–231, 234–235

user-defined subroutines, 56, 57

user-defined variables, 56

users, 146–147, 262

user-submitted data, 7, 9

V

validation, 245, 351

values, 73

variables

- conditional tests on, 197
- configuration, 418
- environment, 5, 8, 197
- loop, 82
- manipulating in subroutines, 90–91
- nesting with references, 100–101
- Perl, 56
- pre-defined (Perl), 56, 376–379

retrieving with SSL, 201

setting, 197, 200

user-defined, 56

VBScript language, 14

Vim text editor, 18

virtual hosting, 20

W

Web browsers

- defined, 2
- displaying dynamic images to, 276–277
- point of view on CGI, 8–9
- receiving data from, 10
- support for, 285

Web servers

- comparing Apache to other, 16–17
- defined, 2
- leasing, 21
- MySQL, 316–317
- point of view on CGI, 10–11
- receiving data from, 9
- sending data to, 9
- support for, 285
- viewing data returned by, 7

Web sites

- ActiveState Downloads, 26
- ActiveState Programmer Network, 24
- adding Facebook Social plugin to, 238–239
- Apache, 23
- Apache Software Foundation, 40
- Artistic License, 24
- CPAN, 108
- developing, 18–19
- enabling Facebook Connect on, 240–243
- Facebook Developer's documentation, 241
- Facebook privacy policy, 243
- forums, 23
- identifying unusual activity on, 350–351
- lighttpd, 17
- nginx, 17
- PerlDoc library, 364
- registering as Facebook applications, 236–237
- Strawberry Perl, 25, 30

while loop, 83–85

Windows

- ActivePerl for, 24, 26–29
- Apache configuration directives, 337

Apache Configuration GUI for, 51
Apache service, 46–47
carriage return character, 55
configuring Apache on, 44–45
creating user directories for Apache in, 130–131
downloading
 Apache for, 40–41
 MySQL for, 320–321
installing
 Apache for, 42–43
 Apache `mod_perl` module for, 358
 MySQL for, 322–323

line feed character, 55
starting and stopping Apache service on, 46–47
Strawberry Perl for, 25, 30–33
[Windows Server, 348–349](#)
[Windows XP, 348](#)
[workflow, 208, 216, 287](#)



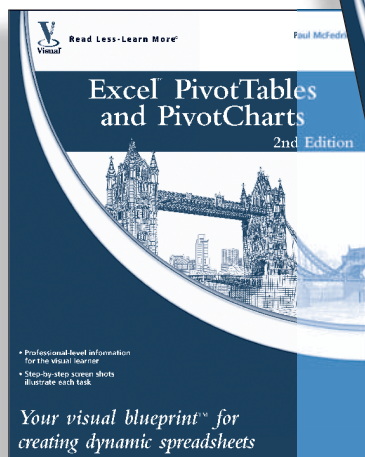
[XML, 296. *See also* AJAX \(Asynchronous JavaScript, and XML\)](#)
[XML::Simple module, 296](#)

For more professional instruction in a visual format, try these.

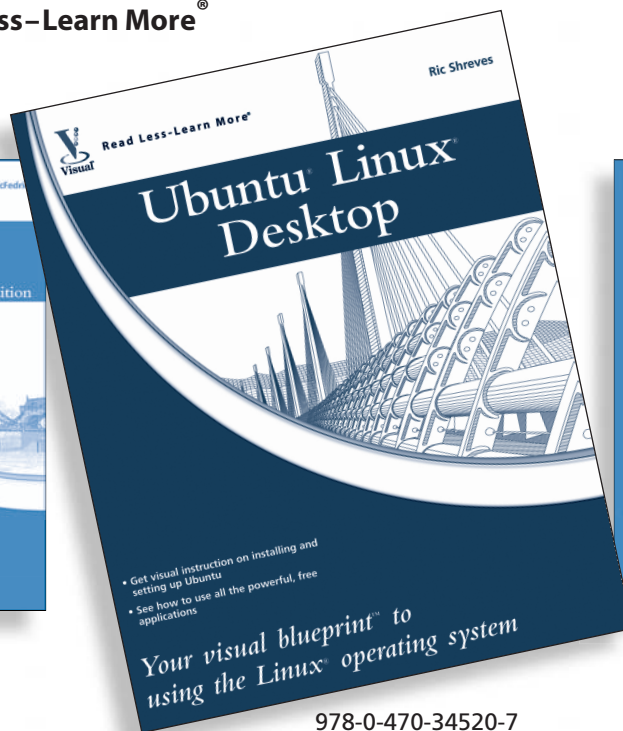
All designed for visual learners—just like you!



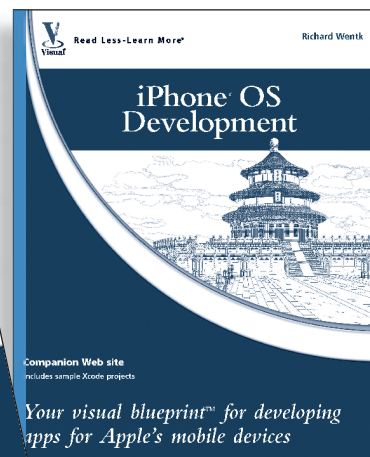
Read Less—Learn More®



978-0-470-59161-1



978-0-470-34520-7



978-0-470-55651-1

For a complete listing of *Visual Blueprint*™ titles and other Visual books, go to wiley.com/go/visual

Wiley, the Wiley logo, the Visual logo, Read Less—Learn More, and Visual Blueprint are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates. All other trademarks are the property of their respective owners.



An Imprint of **WILEY**
Now you know.

Perl and Apache

Welcome to the only guidebook series that takes a visual approach to professional-level computer topics. Open the book and you'll discover step-by-step screen shots that demonstrate over 190 key techniques using Perl and Apache, including:

- Installing Perl and Apache on Linux®
- Building interactive Perl scripts
- Configuring Apache to execute Perl
- Separating HTML code from Perl code
- Processing credit card transactions
- Interfacing a Web site with Facebook®
- Posting status updates to Twitter®
- Creating dynamic images with Perl
- Accessing a back-end MySQL® database
- Securing dynamic Web sites

Enable Facebook Connect on Your Web Site

You can enable Facebook Connect, a single sign-on authentication service provided by Facebook, on your Web site using the Facebook JavaScript SDK and Graph API services. Once you have implemented this feature, a user can instantly be authenticated onto your Web site using their Facebook credentials, provided that they indicate to Facebook that they trust your Web site.

Your Perl CGI template needs to load the Facebook Connect JavaScript SDK, which provides the Facebook Connect Login button. When clicked, the button automatically opens a dialog box that connects to Facebook, prompts the user for their credentials, and sets an access token. Because Facebook Connect exists only as JavaScript code, your Perl script needs to be able to retrieve details about the connected user. The access token, available to your Perl script in the form of a

cookie, can be used to query the Facebook Graph API service to retrieve information about the connected user. The Graph API is the latest Facebook security protocol. Linking it into your Perl CGI code is very easy: you use the LWP::Simple Perl module to query it over HTTPS, and the JSON module to decode its output into a Perl hash reference. Both LWP::Simple and JSON should be pre-installed on most Perl distributions. If they are not, you can install either of them using CPAN, or a platform-specific method as described in Chapter 9. Unfortunately, because Perl is not an officially supported language, this module only supports the older REST API and currently lacks Graph API support. However, manually querying the Graph API is very easy with the help of LWP::Simple and JSON. Additional information is available from the Facebook Authentication documentation at <http://developers.facebook.com/docs/authenticatingperl/>.

Enable Facebook Connect on Your Web Site

- 1 Open a Perl script in a text editor that uses CGI, HTML, Template, and Data-Dumper.
- 2 Type use JSON;
- 3 Type use LWP::Simple;
- 4 Type my \$FacebookAppID = "APP_ID";
- 5 Use the template file facebook-auth.cgi.

- 6 Send your \$FacebookAppID to your template; the JavaScript SDK will use it.
- 7 Retrieve the fb_x-fb-connect cookie.
- 8 Test for the access token value.

Note: Matching regular expression patterns that use brackets will automatically store the matching string in a special variable \$1.

- 9 Type my \$graphURL = "https://graph.facebook.com/";
 - 10 Type my \$json = get(\$graphURL, "me?access_token=\$1");
- Note: The LWP::Simple module provides get(), which is a very easy way to make HTTP or HTTPS queries and forward the results into a Perl scalar.
- 11 Type my \$userData = decode_json(\$json);
- Note: Raw JSON data is very close to a Perl hash in structure, except it does not use any Perl syntax. The function decode_json() makes its data accessible as a hash reference.
- 12 Type \$tmpl->param(\$userData);
 - 13 Type \$tmpl->param(debug => Dumper(\$userData));
- Note: Dumping \$userData is for debugging purposes only. This shows you everything being returned by the Graph API query.
- 14 Type print \$tmpl->output;
 - 15 Save the Perl CGI script.

"I have several books from the Visual series and have always found them to be valuable resources."

— Stephen P. Miller
(Ballston Spa, NY)

- High-resolution screen shots demonstrate each task
- Succinct explanations walk you through step by step
- Two-page lessons break big topics into bite-sized modules
- "Apply It" and "Extra" sidebars highlight useful tips

Extra Apply It

Software Development/General

\$34.99 USA
\$41.99 CAN
£24.99 UK

Visual™
An Imprint of
 WILEY

www.wiley.com/go/visual

ISBN 978-0-470-55680-1
5 34 9 9



9 780470 556801