Ordering Information: **Perl How to Program** & **The Perl Complete Training Course**

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html** To learn more about Deitel instructor-led corporate training delivered at your location, visit **www.deitel.com/training** or contact Christi Kelsey at (978) 461-5880 or e-mail: **christi.kelsey@deitel.net.**

Note from the Authors: This article is an excerpt from Chapter 17, Sections 17.2 and 17.3 of *Perl How to Program, 1/e*. This article introduces LWP and LWP commands. Readers should be familiar with Perl programming, object orientation, basic CGI, file processing and have a basic understanding of HTTP requests and responses.The code examples included in this article show readers examples using the Deitel™ signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

Special Instructions for the code examples:
1. Files **home.html** and **fig16_01.html** should be placed in your Web server's root directory.
2. Program **fig16_02.pl** should be placed in your Web server's **cgi-bin** directory.
3. Program **fig17_01.pl** does not need to be placed in any special directory.
4. When the program runs, the results will be placed in a **response.txt** file, located in the same directory as **fig17_01.pl**.

Web Automation and Networking

17.2 Introduction to LWP

In most client-server interactions, it is the server that sends automated responses to a human using a Web browser on the client computer. Perl has a bundle of modules called ***LWP*** (*Library for the WWW in Perl*), which allows us to automate client-side Web-related activities. Most of the modules in this bundle provide an object-oriented interface. One of the most common uses of LWP is to mimic a browser request for a Web page. We consider some important object-oriented LWP terminology before using the LWP modules.

A *request object* of class **HTTP::Request** contains the information that describes the client's request to the server. The attributes of an **HTTP::Request** are ***method***, ***URL***, ***headers*** and ***content***. Attribute **method** is one of *get*, *put*, *post* or *head*. These will be explained later. Attribute **URL** is simply the address of the requested item. Attribute **headers** is a set of key/value pairs that provide additional information about the request. Attribute **content** contains the data sent from the client to the server as part of the request.

When the server receives a request, it creates a *response object* of the class **HTTP::Response** containing the response to the client. The attributes of an **HTTP::Response** are ***code***, ***message***, ***headers*** and ***content***. Attribute **code** is a status indicator for the outcome of the request. It can represent that the request was success or that an error occurred. Attribute **message** is a string that corresponds to **code**. Attribute **headers** contains additional information about the response and provides a description of the content that the client uses to determine how to process the response. Attribute **content** is the data associated with the response.

Now that we have discussed a request object and a response object, how do we turn the request object into a response object? This is the basis of LWP and is done with a *user agent* object of class ***LWP::UserAgent***. This object acts like the browser in a normal Web interaction—it handles the details of making a request and creating a response object from that request. The response object specifies the response back to the user agent (normally a Web browser). The user agent's primary attributes are ***timeout***, ***agent***, ***from*** and ***credentials***. Attribute **timeout** specifies how long the user agent should wait for a response before timing out (i.e., cancelling the request). Attribute **agent** specifies the name of your user agent. This name will be used when the user agent talks to the network. Attribute **from** is the e-mail address of the person using the Web browser. The attribute **credentials** contains any user names or passwords needed to get a successful response.

17.3 LWP Commands

The program of Fig. 17.1 demonstrates using **LWP** to interact programmatically between a Perl program and a Web server. The output window shows the resulting **response.txt** file that is created by this program. Figure 17.2 displays the contents of **home.html**, one of the files requested in Fig. 17.1. The other file requested, **fig16_02.pl**, is listed in the appedix at the end of this article.

On line 13, we create a new user agent object. On line 14, we create a new request object. The argument to the constructor is a key/value pair indicating that the request is a

GET request and that the URL being requested is **$url** (**'http://localhost/home.html'**). Line 15 calls the user agent's ***request*** method to request the document specified in line 14. The result of this call is stored in a new **HTTP::Response** object. Line 17 calls the **HTTP::Response** object's ***is_success*** method to determine whether the request was successful. If so, line 18 calls the **HTTP::Response** object's ***content*** method and outputs the response to filehandle **OUT**. If the request was not successful, line 21 calls the **HTTP::Response** object's ***status_line*** method to output an error message to filehandle **OUT**. This method returns the status code and message from the response object. In general, if the request is successful, it is unnecessary to view this status code and message.

Good Programming Practice 17.1

When using LWP to grab a page from the Internet, test the HTTP response code for errors to avoid unexpected errors later in the program.

```
1  #!usr/bin/perl
2  # Fig 17.1: fig17_01.pl
3  # Simple LWP commands.
4
5  use strict;
6  use warnings;
7  use LWP::UserAgent;
8
9  my $url = "http://localhost/home.html";
10 open( OUT, ">response.txt" ) or
11    die( "Cannot open OUT file: $!" );
12
13 my $agent = new LWP::UserAgent();
14 my $request = new HTTP::Request( 'GET' => $url );
15 my $response = $agent->request( $request );
16
17 if ( $response->is_success() ) {
18    print( OUT $response->content() );
19 }
20 else {
21    print( OUT "Error: " . $response->status_line() . "\n" );
22 }
23
24 print( OUT "\n----------------------\n" );
25
26 $url = "http://localhost/cgi-bin/fig16_02.pl";
27
28 $request = new HTTP::Request( 'POST', $url );
29 $request->content_type( 'application/x-www-form-urlencoded' );
30 $request->content( 'type=another' );
31 $response = $agent->request( $request );
32
33 print( OUT $response->as_string() );
34 print( OUT "\n" );
35 close( OUT ) or die( "Cannot close out file : $!" );
```

Fig. 17.1 Using **LWP** (part 1 of 2).

```
<html>
<title>This is my home page.</title>

<body bgcolor = "skyblue">
<h1>This is my home page.</h1>
<b>I enjoy programming, swimming, and dancing.</b>
<br></br>
<b><i>Here are some of my favorite links:</i></b>
<br></br>
<a href = "http://www.C++.com">programming</a>
<br></br>
<a href = "http://www.swimmersworld.com">swimming</a>
<br></br>
<a href = "http://www.abt.org">dancing</a>
<br></br></body>
</html>
-----------------------
HTTP/1.1 200 OK
Connection: close
Date: Tue, 21 Nov 2000 15:20:19 GMT
Server: Apache/1.3.12 (Win32)
Content-Type: text/html
Client-Date: Tue, 21 Nov 2000 15:20:19 GMT
Client-Peer: 127.0.0.1:80
Title: Your Style Page

<html><head><title>Your Style Page</title></head>
<body bgcolor = "#ffffc0" text = "#ee82ee"
   link = "#3cb371" vlink = "#3cb371">
<p>This is your style page.</p>
<p>You chose the colors.</p>
<a href = "/fig16_01.html">Choose a new style.</a>
</body></html>
```

Fig. 17.1 Using **LWP** (part 2 of 2).

Next, we try to use a *post* request. This type of request mimics the HTML form element's *post* method. Line 28 creates a new request. The argument to the constructor is a key/value pair indicating that the request is a **POST** request and that the URL being requested is **$url** (**'http://localhost/cgi-bin/fig16_02.pl'**). The URL specified here is that of a CGI program that can handle *post* requests. Line 29 uses the **HTTP::Request** object's ***content_type*** method to set the content type to **'application/x-www-form-urlencoded'**. This specifies how the HTTP protocol will encode the request. Line 30 uses the **HTTP::Request** object's **content** method to specify the data that will be posted to the CGI program. This **content** should be a string that we would send to a *post* request (in this case, **"type=another"**). This information helps the CGI program **fig16_02.pl** create a Web page based on the type it receives. Line 31 executes the request. Line 33 outputs the response to filehandle **OUT**. Note the use of the **HTTP::Response** object's ***as_string*** method, which returns a text representation of the response. This includes some extra information that the **content** method (line 18) does not print.

```
<html>
   <title>This is my home page.</title>

   <body bgcolor = "skyblue">
      <h1>This is my home page.</h1>
      <b>I enjoy programming, swimming, and dancing.</b>
      <br></br>

      <b><i>Here are some of my favorite links:</i></b>
      <br></br>
      <a href = "http://www.C++.com">programming</a>
      <br></br>
      <a href = "http://www.swimmersworld.com">swimming</a>
      <br></br>
      <a href = "http://www.abt.org">dancing</a>
      <br></br>
   </body>
</html>
```

Fig. 17.2 Contents of **`home.html`**.

Another useful request type is a *head* request, which returns only the headers of a request. The headers include information such as the document's type, size and age. This is useful for testing whether a document exists. It is also useful in determining whether the entire document must be downloaded. Web browsers often use such a request to compare the version of a document on the Web server with the version cached on the local computer. If the versions are the same, the Web browser will load the local version for better performance.

Performance Tip 17.1

Using the head *request takes up less time and resources than any of the others. To test whether a page exists, it is almost always better to use a* head *request.*

APPENDIX: LISTING OF `fig16_01.html` *AND* `fig16_02.pl`

Program **`fig16_02.pl`** is a CGI program that displays a Web page with colors specified by the client. Each set of colors is defined as a style. The reader can view the different styles in **`fig16_01.html`**. To run this application on your own, you can simply enter the URL into your Web browser, followed by a query string indicating the desired style. One example is **`http://localhost/cgi-bin/fig16_02?type=dark`**—this query string specifies a darker style for the page.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2  <!-- Fig. 16.1: fig16_01.html                    -->
3  <!-- Web page offering different color options. -->
4
5  <html>
6     <head>
7        <title>Preserving State Through Query Strings</title>
8     </head>
```

Fig. 16.1 HTML document that presents style choices to the user (part 1 of 3).

```
 9
10      <body>
11         <p>Which Style do you Prefer?</p>
12         <table bgcolor = "#ffffff">
13            <tbody>
14               <tr>
15                  <td>
16                     <font color = "#000000">Normal Style</font>
17                  </td>
18                  <td>
19                     <a href = "/cgi-bin/fig16_02.pl?type=normal">
20                     <font color = "#0000ff">Click here</font>
21                  </td>
22               </tr>
23            </tbody>
24         </table>
25         <br/>
26         <table bgcolor = "#dddddd">
27            <tbody>
28               <tr>
29                  <td>
30                     <font color = "#000000">Dark Style</font>
31                  </td>
32                  <td>
33                     <a href = "/cgi-bin/fig16_02.pl?type=dark">
34                     <font color = "#002060">Click here</font>
35                  </td>
36               </tr>
37            </tbody>
38         </table>
39         <br/>
40         <table bgcolor = "#5555ff">
41            <tbody>
42               <tr>
43                  <td>
44                     <font color = "#ee3333">Bright Style</font>
45                  </td>
46                  <td>
47                     <a href = "/cgi-bin/fig16_02.pl?type=bright">
48                     <font color = "#ffff00">Click here</font>
49                  </td>
50               </tr>
51            </tbody>
52         </table>
53         <br/>
54         <table bgcolor = "#ffffc0">
55            <tbody>
56               <tr>
57                  <td>
58                     <font color = "#ee82ee">Another Style</font>
59                  </td>
60                  <td>
61                     <a href = "/cgi-bin/fig16_02.pl?type=another">
```

Fig. 16.1 HTML document that presents style choices to the user (part 2 of 3).

```
                    <font color = "#3cb371">Click here</font>
                </td>
            </tr>
        </tbody>
    </table>
    <br/>
  </body>
</html>
```

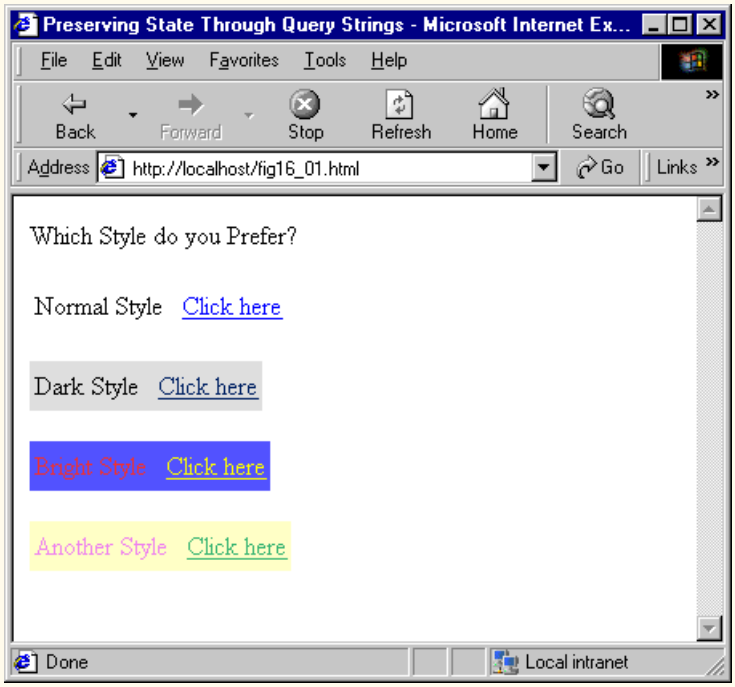

Fig. 16.1 HTML document that presents style choices to the user (part 3 of 3).

```
#!/usr/bin/perl
# Fig. 16.2: fig16_02.pl
# Create a page with a specified style.

use strict;
use warnings;
use CGI qw( :standard );

print( header() );

my $type = param( "type" );

```

Fig. 16.2 Perl CGI program that creates a Web page that uses the style selected by the user in Fig. 16.1 (part 1 of 2).

```
13  my %colors = ( "normal" => [ "#ffffff", "#000000", "#0000ff" ],
14     "dark" => [ "#dddddd", "#000000", "#002060" ],
15     "bright" => [ "#5555ff", "#ee3333", "#ffff00" ],
16     "another" => [ "#ffffc0", "#ee82ee", "#3cb371" ] );
17
18  my $style = $colors{ $type };
19  my @style = @{ $style };
20
21  print <<HTML;
22  <html><head><title>Your Style Page</title></head>
23  <body bgcolor = "$style[ 0 ]" text = "$style[ 1 ]"
24     link = "$style[ 2 ]" vlink = "$style[ 2 ]">
25  <p>This is your style page.</p>
26  <p>You chose the colors.</p>
27  <a href = "/fig16_01.html">Choose a new style.</a>
28  </body></html>
29  HTML
```

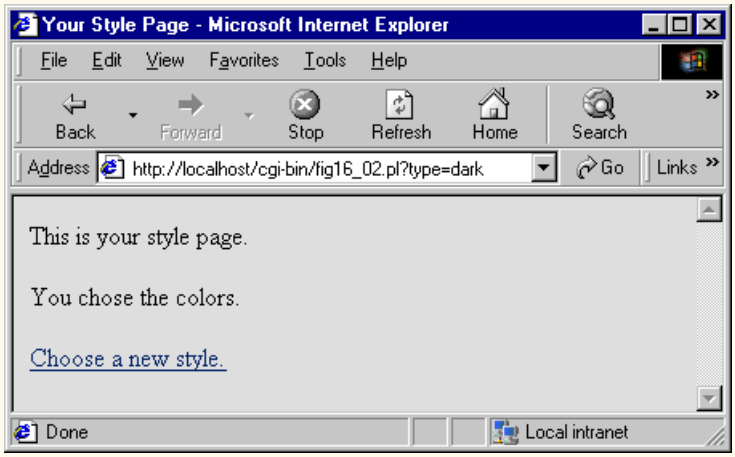

Fig. 16.2 Perl CGI program that creates a Web page that uses the style selected by the user in Fig. 16.1 (part 2 of 2).