Developing Dynamic Web Pages: Part 1

by Steve Troxell

Last month, Bob Swart's *Under Construction* column gave you an introduction to using Delphi 2 to create an Internet server application that processes a query request from a browser through the Common Gateway Interface (CGI). In this article, we'll complete the discussion and look at some of the different forms of CGI and other methods. Next month, we'll see how to implement database access by including the BDE.

To recap the premise of CGI apps from Bob's article, a web page might be designed to contain one or more data entry controls within a container space called a "form" on the HTML page (I'll call this the "query page"). The user fills in the information for these controls from their browser and then "submits" the form contents (that is, the values for the data entry controls) to the web server, usually by clicking a Submit button on the query page. This sends a special HTTP message to the web server which includes the data values for all the data controls on the query page as well as the location and name of a dedicated server-based program or script to process the request. This program is identified as part of the HTML definition for the query page.

This program's job is to read the data values sent by the client, do some sort of processing, and return a formatted HTML page back to the client (I'll call this the response page). Presumably, the response page contains information customized in accordance with whatever data values the user had entered on the original query page.

Standard CGI

The technique Bob introduced to you is called "standard CGI" and was developed by the National Center for Supercomputing Applications (the folks who created the WWW). Standard CGI is a de facto standard for processing client input received from a web browser and, as such, is very widely available to most web servers on most platforms.

As you saw last month, the web server communicates the parameters of the CGI session to the CGI program via environment variables (see Figure 1) and generally provides the user input via the standard input device. The program merely reads and decodes text records from the standard input device and produces a response page by writing HTML formatted text to the standard output device.

In reality, the mechanism by which your CGI app receives the user input – the contents of the data controls from the query page – varies depending on the type of "submit" action defined in the query page. When a user clicks the Submit button on a query page, the action may be a GET or a POST. When the submit action is POST, as in Bob's case last month, the user input is indeed passed through to the CGI program via the standard input device. However, when the submit action is GET, the user input is passed via the QUERY_STRING environment variable. Processing forms through GET may be a bit faster than POST because no file I/O is involved to read the user data. However, you're restricted to the length of that operating system's environment variable for passing all your form variables to the CGI application. The data received in either case is formatted identically, there are just two different pipelines from which you might obtain it. We can determine which action we are processing by examining the REQUEST METHOD variable.

WinCGI

The standard input/output model for CGI arose from the UNIX platforms which were widely used in the early days of the web (and still are to some extent). It was fairly straightforward for the web development tools available for UNIX, chiefly C and Perl, to manipulate these devices. In recent years, Windows servers have become more and more popular as web servers. While most Windows development tools are quite capable of dealing with the standard input and output devices, some (such as Visual Basic) are not.

WinCGI is an alternative CGI interface originally developed by Robert Denny for his WebSite web server software but now supported by other vendors as well. WinCGI uses a Windows INI file to pass CGI session parameters and user data

➤ *Figure 1*

```
SERVER_SOFTWARE=Microsoft-Internet-Information-Server/1.0
SERVER_NAME=www.stevet.tpower.com
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.0
SERVER_PORT=80
REQUEST_METHOD=POST
PATH_TRANSLATED=C:\INetPub\WWWROOT
SCRIPT_NAME=/cgi-bin/cgitest.exe
REMOTE_HOST=163.225.101.7
REMOTE_ADDR=163.225.101.7
CONTENT_TYPE=application/x-www-form-urlencoded
CONTENT_LENGTH=113
HTTP_USER_AGENT=Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

into the CGI application. Since INI files are widely accessible to any development platform for Windows, WinCGI became popular.

The WinCGI program receives a single parameter on the command line which identifies the CGI data file: an INI-format file containing all the values you would have obtained through environment variables with standard CGI (see Figure 2). All the values for the form's variables are found in the `[Form Literal]` section, already parsed and decoded. Rather than write the response page to the standard output device, we write to the file specified by the `Output File` key in the `[System]` section. The server will then take this file and pass it back to the client. Note the subtle differences in syntax from standard CGI variable names: keywords are separated by spaces rather than underscores and some of the names are slightly different.

Server APIs

The biggest problem with standard CGI and Windows CGI is that they are slow. These CGI applications run as separate processes for each request. That is, each time they are accessed, the CGI program is loaded, initialized, executed and shut down. Because these CGI processes are not persistent, this startup/shutdown overhead for each request is the most significant drawback to CGI programs: they're slow to process requests.

Web server APIs, such as ISAPI and NSAPI, allow more direct processing of server requests than is possible with normal CGI methods. Usually, you'll write a DLL to handle server requests through the server API. As such, the DLL runs within the web server process space. Because of this, API techniques provide much faster processing of server requests than traditional CGI.

Server APIs also allow you to write server request filters and authenticators. For example, a filter might be used to append a sponsor's advertising within certain web pages accessed by the browser. An authenticator can control access to certain parts of the web site by requiring a username and password and comparing them against a database of access rights before returning the requested page.

However, the use of server APIs also carries with it the prospect that a buggy DLL could corrupt the entire web server and they are inherently proprietary to the web server being used. Because of this, it may be better to get experience building dynamic web content through CGI, even if ultimately you want the speed and power of server APIs. Also, before you count CGI out, take a look at...

FastCGI

FastCGI, offered by Open Market Inc and endorsed by the NCSA, is a new improvement over traditional CGI. FastCGI performs the same tasks as CGI, but does so from a persistent, isolated process that listens on a socket connection for requests to process. Since FastCGI runs in a separate process space from the web server itself, there is no startup/shutdown overhead with each request and risks to the server from buggy applications are reduced. FastCGI promises the safety and ease of programming of CGI with the performance and flexibility of native web server APIs. Also, programs written for CGI will be compatible with FastCGI servers. We will not cover FastCGI here since it is not yet widely available, but support is planned for all the major web servers including Microsoft and Netscape. Watch this space!

Data Encoding

Back to the work at hand. We're going to focus on standard CGI and WinCGI for the remainder of this article. Before we go any further, we need to clarify how data is encoded with these methods.

User data handed over to a program by standard CGI or WinCGI

➤ *Figure 2*

```
[CGI]
Request Method=POST
Query String=
Logical Path=
Physical Path=C:\INetPub\WWWROOT
CGI Version=CGI/1.2 (Win)
Request Protocol=HTTP/1.0
Executable Path=/cgi-bin/project2.dll
Server Software=Microsoft-IIS/2.0
Server Name=www.stevet.tpower.com
Server Port=80
Remote Host=163.225.101.7
Remote Address=163.225.101.7
Referer=http://www.steve.tpower.com/is2wcgi.htm
Content File=P:\WINNT\key6F.tmp
Content Length=64
Content Type=application/x-www-form-urlencoded

[Accept]
image/gif=Yes
image/x-xbitmap=Yes
image/jpeg=Yes
image/pjpeg=Yes
*/*=Yes

[System]
GMT Offset=-25200
Debug Mode=No
Output File=C:\INetPub\WWWRoot\cgi-bin\CGI6E.tmp
Content File=C:\WINNT\key6F.tmp

[Extra Headers]
ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
ACCEPT-LANGUAGE=en
CONNECTION=Keep-Alive
CONTENT-LENGTH=64
CONTENT-TYPE=application/x-www-form-urlencoded
HOST=www.steve.tpower.com
REFERER=http://www.steve.tpower.com/is2wcgi.htm
UA-PIXELS=800x600
UA-COLOR=color8
UA-OS=Windows 95
UA-CPU=x86
USER-AGENT=Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)
PRAGMA=No-Cache
EXTENSION=Security/Remote-Passphrase

[Form Literal]
Text1=sample text
TextWithDefault=Default
Check1=
Dropdown1=John
```

through the `QUERY_STRING` variable or the standard input device is URL-encoded (the user data file passed in from WinCGI via the `Content File` key is already decoded by the server). As such, your programs must be prepared to decode this data. As Bob pointed out, user data is sent in `keyword=value` pairs separated by an ampersand.

Since the equals sign and the ampersand are used as special delimiters for the keyword/value pairs, what happens if the user enters one of these characters? In this event, escape sequences are used to represent the characters. Escape sequences consist of the `%` character followed by a 2-digit hexadecimal number which is the ASCII value of the character. Therefore, if `=` or `&` appear literally within the user data, they are converted to `%3D` and `%26` respectively. Likewise, if a literal percent sign is part of the user data, then it is converted to `%25`.

Finally, any spaces in the user data are converted to plus signs. Bear in mind that if plus signs are inherently present in the user data, they are converted to the escape sequence `%2B`. Any literal plus sign in the user data can therefore be treated as a space. [**Editor's note**: In the November issue Bob Swart implied that in the *URL encoding* spaces were replaced by underscores. This was actually a replacement that Bob's code was carrying out in both the CGI form and the Web-hosted book reviews database file.]

Another thing to remember is that for extended ASCII characters, the actual character intended by the escape sequence depends on the character set (code page) used on the client computer.

Figure 3 shows an example of user input in a query form with three data controls and the corresponding URL-encoded user data received by the CGI program.

Data Control	**User Enters**
AUTHOR	John & Jane Doe
TITLE	C++ is for Losers
ROYALTY	10%

URL-Encoded Text

AUTHOR=John+%26+Jane+Doe&TITLE=C%2B%2B+is+for+Losers&ROYALTY=10%25

➤ *Figure 3*

The TCGI Component

Now we're going to encapsulate all that Bob taught us last month about standard CGI and all we know about WinCGI into a single component that will allow our programs to work transparently with either protocol. The `TCGI` component reads the CGI environment and passes the information on to the program through a standard interface. Our program reads the values out of the `TCGI` component rather than interfacing directly with the CGI implementation on the server.

The `TCGI` component is shown in Listing 1. This component surfaces two properties for accessing CGI information: `CGIItems` and `FormItems`. These are `TStringList` components which contain `keyword=value` pairs for all CGI variables and form variables respectively.

Since the naming of CGI variables differs slightly between standard CGI and WinCGI, we must decide on a single naming convention for both. For our purposes here, we will use the WinCGI variable names. The names we assign for common access are defined in the `CGIVarNames` array. These are the names we will use when referencing CGI variables from our applications. The actual variable names used by each protocol are defined in the `CGIVars` array. Form variable names are passed through as-is under both protocols, so we always refer to them by the names we assigned in the HTML definition on the query page.

The heart of the `TCGI` component is its `Create` method. Since the CGI data really only moves one way (from the browser to the application), there's no reason not to load up everything immediately. The Create method first determines whether we're operating in a standard CGI or WinCGI environment. This is crucial to all the other operations in the component. Next, the output file (which may be either standard output or a temporary file) is opened for writing the response page and the required header lines are written for us. Finally, we load up the CGI variables into the `CGIItems` component and the form variables into the `FormItems` component.

The technique for loading the form variables varies greatly between the two protocols, so I decided to break that functionality out into two separate methods. For standard CGI, we must decide if the form action is a `POST` or a `GET`. On `POST`s, we read and decode `CONTENT LENGTH` bytes from standard input, just as Bob showed us last month. For `GET`s, we must read and decode the contents of the `QUERY STRING` variable.

Doing this same task for WinCGI is vastly simpler since the server has already done the hard work for us. All we have to do is copy the `[Form Literal]` section of the INI file into our `FormItems` component. Luckily, `TIniFile.ReadSectionValues` was tailor-made for this.

Testing The TCGI Component

Listing 2 shows a test CGI program which simply dumps all the CGI input into an HTML response page. Figure 4 and Listing 3 show a test HTML page that calls `CGITest` to exercise our component. This test assumes that the HTML query page (TEST.HTM) and the CGITEST.EXE executable are stored in the same directory on the web server.

TEST.HTM reveals two HTML forms with sample data controls: one form to test standard CGI and one form to test WinCGI. Two forms are rarely used in real life, but we did so here because we need two different `ACTION` parameters to exercise the two forms of

```
unit CGIAPI;
interface
uses Classes, INIFiles;
type
  TEnvironmentType = (etStdCGI, etWinCGI);
  TCGI = class
  private
    FCGIItems: TStringList;
    FFormItems: TStringList;
    FEnvironmentType: TEnvironmentType;
    FOutputFile: TextFile;
    FWinCGIProfileName: string;
    FWinCGIProfile: TINIFile;
  protected
    constructor Create; virtual;
    destructor Destroy; override;
    procedure LoadStdCGIUserData;
    procedure LoadWinCGIUserData;
    procedure UnpackURLString( S: PChar ); virtual;
  public
    procedure DumpWinCGIProfile;
    function GetEnv(Variable: string): string;
    procedure Write(Value: string);
    procedure WriteLn(Value: string);
    property CGIItems: TStringList read FCGIItems;
    property EnvironmentType: TEnvironmentType
      read FEnvironmentType;
    property FormItems: TStringList read FFormItems;
    property OutputFile: TextFile read FOutputFile
      write FOutputFile;
    property WinCGIProfile: TINIFile read FWinCGIProfile;
  end;
var CGI: TCGI;
implementation
uses SysUtils, Windows;
const
  NumCGIVars = 15;
  { Standard names used by calling application to reference
    CGI variables. They generally follow the WinCGI names. }
  CGIVarNames: array[0..NumCGIVars - 1] of string[31] =
    ('SERVER SOFTWARE',     'SERVER NAME',
     'SERVER PORT',         'CGI VERSION',
     'REQUEST PROTOCOL',    'REQUEST METHOD',
     'LOGICAL PATH',        'PHYSICAL PATH',
     'EXECUTABLE PATH',     'QUERY STRING',
     'REMOTE HOST',         'REMOTE ADDRESS',
     'REMOTE USER',         'CONTENT LENGTH',
     'CONTENT TYPE');
  { These are actual variable names used by each protocol. }
  CGIVars: array[0..NumCGIVars - 1, TEnvironmentType] of
    string[31] =
     { etStdCGI              etWinCGI }
    (('SERVER_SOFTWARE',    'SERVER SOFTWARE'),
     ('SERVER_NAME',        'SERVER NAME'),
     ('SERVER_PORT',        'SERVER PORT'),
     ('GATEWAY_INTERFACE',  'CGI VERSION'),
     ('SERVER_PROTOCOL',    'REQUEST PROTOCOL'),
     ('REQUEST_METHOD',     'REQUEST METHOD'),
     ('PATH_INFO',          'LOGICAL PATH'),
     ('PATH_TRANSLATED',    'PHYSICAL PATH'),
     ('SCRIPT_NAME',        'EXECUTABLE PATH'),
     ('QUERY_STRING',       'QUERY STRING'),
     ('REMOTE_HOST',        'REMOTE HOST'),
     ('REMOTE_ADDR',        'REMOTE ADDRESS'),
     ('REMOTE_USER',        'REMOTE USER'),
     ('CONTENT_LENGTH',     'CONTENT LENGTH'),
     ('CONTENT_TYPE',       'CONTENT TYPE'));
constructor TCGI.Create;
var I: Integer;
begin
  inherited Create;
  FCGIItems := TStringList.Create;
  FFormItems := TStringList.Create;
  { Detect whether we are standard CGI or WinCGI. }
  if GetEnv('SERVER_NAME') <> '' then
    FEnvironmentType := etStdCGI
  else begin
    FEnvironmentType := etWinCGI;
    FWinCGIProfileName := ParamStr(1);
    FWinCGIProfile := TINIFile.Create(FWinCGIProfileName);
  end;
  { Assign and open our output file accordingly. }
  case EnvironmentType of
    etStdCGI: AssignFile(OutputFile, '');
    etWinCGI: AssignFile(OutputFile,
      WinCGIProfile.ReadString('System', 'Output File', ''));
  end;
  Rewrite(OutputFile);
  { Write standard HTML header for the output page. }
  WriteLn('Content-type: text/html');
  WriteLn('');
  { Load CGI variables and user's form variables. }
  case EnvironmentType of
    etStdCGI: begin
                for I := 0 to NumCGIVars - 1 do
                  FCGIItems.Values[CGIVarNames[I]] :=
                    GetEnv(CGIVars[I, etStdCGI]);
                LoadStdCGIUserData;
              end;
    etWinCGI: begin
                for I := 0 to NumCGIVars - 1 do
                  FCGIItems.Values[CGIVarNames[I]] :=
                    WinCGIProfile.ReadString('CGI',
                      CGIVars[I, etWinCGI], '');
                LoadWinCGIUserData;
              end;
  end;
end;
destructor TCGI.Destroy;
begin
  CloseFile(OutputFile);
  FCGIItems.Free;
  FFormItems.Free;
  FWinCGIProfile.Free;
end;
procedure TCGI.DumpWinCGIProfile;
{ Writes contents of WinCGI profile file to response page. }
var FCB: TextFile;
    Rec: string;
begin
  if FWinCGIProfile <> nil then begin
    AssignFile(FCB, FWinCGIProfileName);
    Reset(FCB);
    try
      while not Eof(FCB) do begin
        ReadLn(FCB, Rec);
        WriteLn(Rec + '<BR>');
      end;
    finally
      CloseFile(FCB);
    end;
  end;
end;
function TCGI.GetEnv(Variable: string): string;
{ Returns the value of the given environment variable. }
var EnvVariable: array[0..127] of char;
    EnvBuffer: array[0..1023] of char;
begin
  StrPCopy(EnvVariable, Variable);
  Result := '';
  if GetEnvironmentVariable(PChar(Variable), @EnvBuffer,
    SizeOf(EnvBuffer)) <> 0 then
    Result := StrPas(EnvBuffer);
end;
procedure TCGI.LoadStdCGIUserData;
{ Reads, parses, decodes values for standard CGI form variables }
var
  ContentLength: LongInt;
  InputFCB: File;
  InputBuffer: PChar;
  RequestMethod: string;
  UserContentBuffer: string;
begin
  RequestMethod :=
    Uppercase(FCGIItems.Values['REQUEST METHOD']);
  { If form action is a POST, then get form variables
    from standard input device. }
  if RequestMethod = 'POST' then begin
    if FCGIItems.Values['CONTENT TYPE'] <> '' then begin
      ContentLength :=
        StrToInt(FCGIItems.Values['CONTENT LENGTH']);
      AssignFile(InputFCB, '');  { standard input }
      Reset(InputFCB, 1);
      try
        InputBuffer := StrAlloc(ContentLength + 1);
        FillChar(InputBuffer^, ContentLength + 1, #0);
        try
          BlockRead(InputFCB, InputBuffer^, ContentLength);
          UnpackURLString(InputBuffer);
        finally
          StrDispose(InputBuffer);
        end;
      finally
        CloseFile(InputFCB);
      end;
    end;
  end
  { If the form action is GET, then we get form variables
    from the QUERY STRING variable. }
  else if RequestMethod = 'GET' then begin
    UserContentBuffer := FCGIItems.Values['QUERY STRING'];
    InputBuffer := StrAlloc(Length(UserContentBuffer));
    try
      StrPCopy(InputBuffer, UserContentBuffer);
      UnpackURLString(InputBuffer);
    finally
      StrDispose(InputBuffer);
    end;
  end;
end;
procedure TCGI.LoadWinCGIUserData;
{ Copies values for WinCGI form variables. }
begin
  { All form variables are found in the [Form Literal]
    section of the profile file. }
  WinCGIProfile.ReadSectionValues('Form Literal',
    TStrings(FFormItems));
end;
procedure TCGI.UnpackURLString( S: PChar );
{ Parses and decodes a URL-encoded string. Copies variable
  values into the FFormItems field. }
{ Listing continues on facing page... }
```

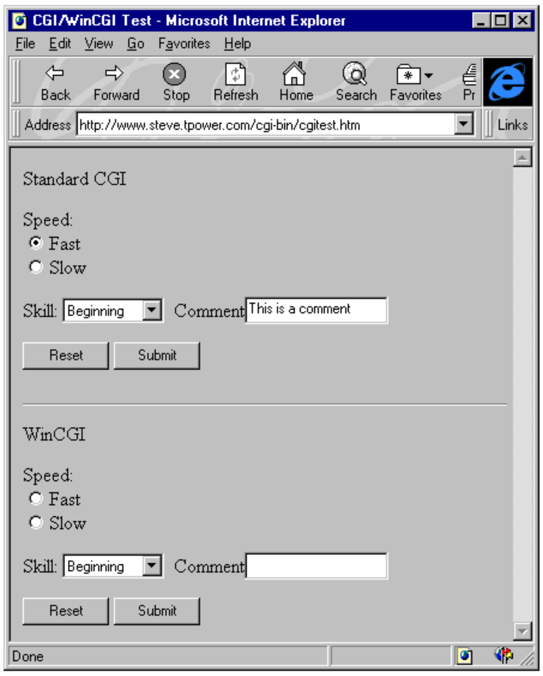
```
{Listing continued from facing page... }

var LabelStr: ShortString;
    ValueStr: ShortString;
begin
  LabelStr := '';
  ValueStr := '';
  while S^ <> #0 do begin
    case S^ of
      '+' : ValueStr := ValueStr + ' ';
      '%' : begin
              ValueStr := ValueStr +
                Chr(StrToInt('$' + (S + 1)^ + (S + 2)^));
              Inc(S, 2);
            end;
      '=' : if LabelStr = '' then begin
              LabelStr := ValueStr;
              ValueStr := '';
            end;
      '&' : begin
              FFormItems.Values[LabelStr] := ValueStr;
              ValueStr := '';
              LabelStr := '';
            end;
      else ValueStr := ValueStr + S^;
    end;
    Inc(S);
  end;
  if ValueStr <> '' then
    FFormItems.Values[LabelStr] := ValueStr;
end;
procedure TCGI.Write(Value: String);
{ Standard Write to the output page. }
begin
  System.Write(OutputFile, Value);
end;
procedure TCGI.WriteLn(Value: String);
{ Standard WriteLn to the output page. }
begin
  System.WriteLn(OutputFile, Value);
end;
initialization
  CGI := TCGI.Create;
finalization
  CGI.Free;
end.
```

➤ *Facing page and above: Listing 1*

➤ *Right: Figure 4*

```
program CGITest;
{$APPTYPE CONSOLE}
uses SysUtils, CGIAPI;
var I: Integer;
begin
  with CGI do begin
    WriteLn('<HTML><HEAD>');
    WriteLn('<TITLE>CGI/WinCGI Test Page</TITLE>');
    WriteLn('</HEAD><BODY>');
    Write('Environment: ');
    case EnvironmentType of
      etStdCGI: WriteLn('Standard CGI<BR>');
      etWinCGI: WriteLn('WinCGI<BR>');
    end;
    WriteLn('<BR>');
    WriteLn('CGI Variables:<BR>');
    for I := 0 to CGI.CGIItems.Count - 1 do
      WriteLn(CGI.CGIItems[I] + '<BR>');
    WriteLn('<BR>');
    WriteLn('Form Variables:<BR>');
    for I := 0 to CGI.FormItems.Count - 1 do
      WriteLn(CGI.FormItems[I] + '<BR>');
    WriteLn('</BODY></HTML>');
  end;
end.
```

➤ *Listing 2*


CGI/WinCGI Test - Microsoft Internet Explorer

Address: http://www.steve.tpower.com/cgi-bin/cgitest.htm

Standard CGI

Speed:
(•) Fast
( ) Slow

Skill: Beginning   Comment: This is a comment

Reset   Submit

WinCGI

Speed:
( ) Fast
( ) Slow

Skill: Beginning   Comment:

Reset   Submit

Done

For Further Information

CGI:	hoohoo.ncsa.uiuc.edu/cgi/
	www.eborcom.com/webmaker/#cgip
	http://ute.usi.utah.edu/bin/cgi-programming/counter.pl/ cgi-programming/index.htm
WinCGI:	http://website.ora.com/wsdocs/32demo/windows-cgi.html
FastCGI:	www.fastcgi.com

CGI. The calling convention shown for the WinCGI `ACTION` property is how you emulate WinCGI with Microsoft Internet Information Server. The CGITEST.DLL file resides in the same directory as the TEST.HTM page. CGITEST.DLL is actually a renamed copy of IS2WCGI.DLL obtained from the Win32 SDK for Windows NT 4.0. When this DLL is invoked, it translates the standard CGI environment variables into WinCGI INI files and executes an EXE of the same name in the same directory. Remember, we have only one CGITEST.EXE for either standard CGI or WinCGI: there are no coding changes required.

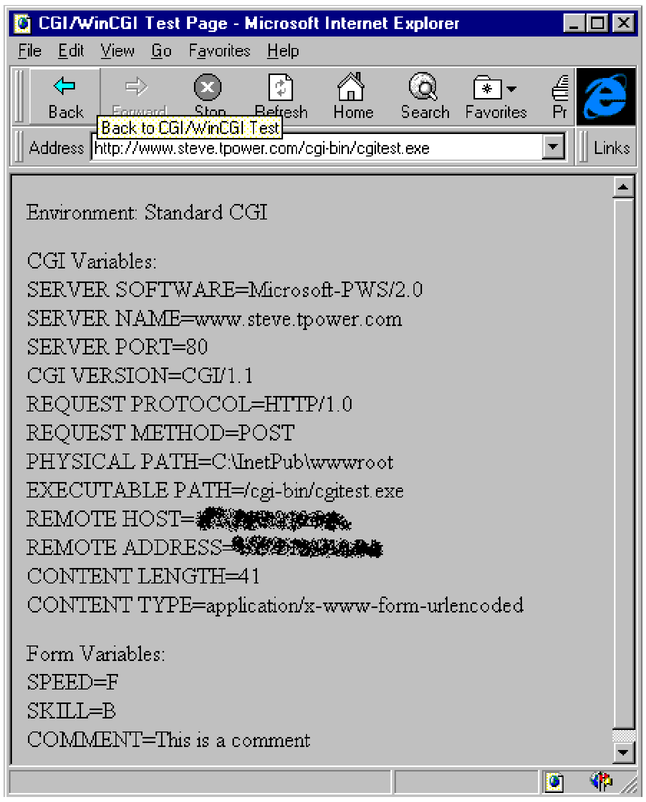
Figures 5 and 6 (next page) show the response pages for standard CGI and WinCGI respectively.

Conclusion

The TCGI component is not meant to be a commercial-grade Internet component. This was more of an educational exercise than anything. You can expand on it as much as you like and all your CGI apps could then share common access through your version of `TCGI`. Also, bear in mind that you can still access everything through the `GetEnv` function and the `WinCGIProfile` property, although you would no longer be transparent to which CGI protocol was being used.

In the next issue, we'll embellish the TCGI component a bit and see how to use it to build the web version of The Delphi Magazine's Article Index Database (with BDE access and everything!).

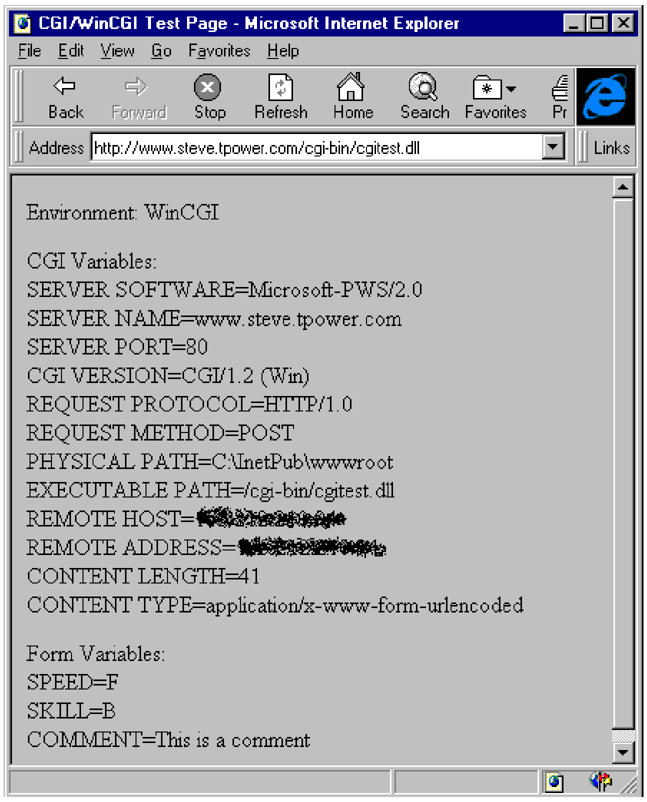
Steve Troxell is a Senior Software Engineer with TurboPower Software. He can be reached by email at stevet@tpower.com or on CompuServe at 74071,2207


CGI/WinCGI Test Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Pr
Back to CGI/WinCGI Test
Address
Links

http://www.steve.tpower.com/cgi-bin/cgitest.exe

```
Environment: Standard CGI

CGI Variables:
SERVER SOFTWARE=Microsoft-PWS/2.0
SERVER NAME=www.steve.tpower.com
SERVER PORT=80
CGI VERSION=CGI/1.1
REQUEST PROTOCOL=HTTP/1.0
REQUEST METHOD=POST
PHYSICAL PATH=C:\InetPub\wwwroot
EXECUTABLE PATH=/cgi-bin/cgitest.exe
REMOTE HOST=[redacted]
REMOTE ADDRESS=[redacted]
CONTENT LENGTH=41
CONTENT TYPE=application/x-www-form-urlencoded

Form Variables:
SPEED=F
SKILL=B
COMMENT=This is a comment
```

➤ *Figure 5 (IP addresses deliberately distorted)*


CGI/WinCGI Test Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Pr
Address
Links

http://www.steve.tpower.com/cgi-bin/cgitest.dll

```
Environment: WinCGI

CGI Variables:
SERVER SOFTWARE=Microsoft-PWS/2.0
SERVER NAME=www.steve.tpower.com
SERVER PORT=80
CGI VERSION=CGI/1.2 (Win)
REQUEST PROTOCOL=HTTP/1.0
REQUEST METHOD=POST
PHYSICAL PATH=C:\InetPub\wwwroot
EXECUTABLE PATH=/cgi-bin/cgitest.dll
REMOTE HOST=[redacted]
REMOTE ADDRESS=[redacted]
CONTENT LENGTH=41
CONTENT TYPE=application/x-www-form-urlencoded

Form Variables:
SPEED=F
SKILL=B
COMMENT=This is a comment
```

➤ *Figure 6 (IP addresses deliberately distorted)*

```
<HTML>
<HEAD><TITLE>CGI/WinCGI Test</TITLE></HEAD>
<BODY>
<P>
Standard CGI<BR>
<BR>
<FORM ACTION=
  “http://www.steve.tpower.com/cgi-bin/cgitest.exe”
  METHOD="POST">
Speed:<BR>
<INPUT TYPE="radio" NAME="SPEED" VALUE="F" CHECKED>Fast<BR>
<INPUT TYPE="radio" NAME="SPEED" VALUE="S"        >Slow<BR>
<P>
Skill:
<SELECT NAME="SKILL">
  <OPTION VALUE="B"> Beginning
  <OPTION VALUE="I"> Intermediate
  <OPTION VALUE="A"> Advanced
</SELECT>
Comment<INPUT TYPE="text" NAME="COMMENT">
<P>
<INPUT TYPE="RESET">
<INPUT TYPE="SUBMIT">
</FORM>
<HR>


<P>
WinCGI<BR>
<BR>
<FORM ACTION=
  “http://www.stevet.tpower.com/cgi-bin/cgitest.dll”
  METHOD="POST">
Speed:<BR>
<INPUT TYPE="radio" NAME="SPEED" VALUE="F" CHECKED>Fast<BR>
<INPUT TYPE="radio" NAME="SPEED" VALUE="S"        >Slow<BR>
<P>
Skill:
<SELECT NAME="SKILL">
  <OPTION VALUE="B"> Beginning
  <OPTION VALUE="I"> Intermediate
  <OPTION VALUE="A"> Advanced
</SELECT>
Comment<INPUT TYPE="text" NAME="COMMENT">
<P>
<INPUT TYPE="RESET">
<INPUT TYPE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

➤ *Listing 3*