Under Construction:
A Delphi 2.01 CGI Debugger

by Bob Swart

In Issue 15, we started to explore the internet and specifically the world wide web. Our goal back then was to put the search engine for The Delphi Magazine Review Database of Delphi Books (TDMBKS) onto the web, including the individual reviews themselves. We used Delphi 2 to implement a standard CGI application. In the following issues, Steve Troxell explained the difference between standard CGI and WinCGI and showed us how to implement The Delphi Magazine Article Index Database, TDMAid, using WinCGI. In this issue, Steve goes on to discuss developing web applications using ISAPI.

Writing a CGI application is one thing. Testing and debugging it is quite another. A CGI application runs on a Web Server (such as IIS), so you cannot test it with only a Web Browser (such as Netscape Navigator). This simple fact can seriously slow down the speed of development of your CGI application, as you need to upload it to your web server for each little change you've made (and this means a phone call to your ISP or at least an ftp-connection to your web server). As an alternative, you can set up an Intranet on your LAN, or buy and install Microsoft FrontPage, which has a Personal WebServer that supports CGI (note: only the official version supports CGI applications, the latest beta does not), but I'd rather not spend money on some tool I'll only be using a small part of. Besides, it's not invented here!

In order to enhance the flexibility of Delphi CGI development, this month we're going to construct a small (simple and somewhat limited) Web Browser which can also be used to configure and execute our CGI applications and show us the results.

Internet Solutions Pack

For those of you who haven't upgraded to Delphi 2.01 yet, here's a reason to do so: it includes the Internet Solutions Pack, a free set of (beta) internet ActiveX controls from NetManage. Sure, they're beta, and you need a patch to make sure they continue working after June 1997, but even as little toys you can use them to make a few much nicer and bigger toys. Moreover, we're gonna need these controls (or at least the HTML ActiveX control) for the remainder of this article, so do yourself a favour and upgrade now!

The NetManage ActiveX internet solutions pack consists of eight ActiveX components in four categories: basic winsock controls (TCP and UDP), protocol client controls (FTP, HTTP, NNTP, POP3 and SMTP), infrastructure controls and COM objects (internal classes), and finally web controls (HTML).

At first sight, this may seem a wonderful addition to 32-bit Delphi and Borland C++Builder, as it opens up the way to the web and internet for every user. However, there are some limitations as well, such as the fact that there's no paper documentation provided for these components (and the on-line help just doesn't!). The only information available is in the help files provided by NetManage and the few examples on Borland's web site. For our CGI Debugger, we'll explore the HTML ActiveX control, so at the end of this article at least this component should have a few less secrets for you.

HTML

If you drop the HTML ActiveX control on a form and press F1 then you get the general help for the Internet Solution Pack components. This general help page is also the entrance to the individual components' help files. The HTML ActiveX control is a visual component which you can use to view or parse HTML pages, and it supports the following features:

- HTML version 2.x plus most NetScape 2.0 and Explorer 2.0 extensions (but nothing for 3.0);
- Inline graphics: GIF, JPEG, BMP, XBM;
- Built-in document retrieval for HTTP and File URLs;
- Built-in HTTP form execution.

Note that it only supports HTML version 2.x, which means that frames are not possible! (a pity, since my home page and HTML Experts depend a lot on frames). Tables are supported, however.

Web Browser

To illustrate the initial ease of using the HTML ActiveX control, let's do again what was first shown at the BDC'96 in Anaheim: build a small web browser in a few lines of code. All we need is a ComboBox, a Button, an HTML ActiveX control and (optionally) a StatusBar. Drop them on a Form (or the first page of a PageControl). We can pre-fill the ComboBox with a list of possible sites (our favourite bookmarks, for example) or just enter a new URL into it. Whenever we click on the button, the URL is given to the HTML ActiveX control, which is then requested to get the document as shown in the `JumpBtnClick` method in Listing 1.

The StatusBar here consists of two panels: one for the OK/BAD status of the URL and the second one for a longer message, such as the URL itself or the connection process. If we want to show the URL of the document which is currently being retrieved in the second panel of the StatusBar, we can use the `OnDoRequestDoc` event handler of the HTML ActiveX control

for this as shown in HTML1DoRequestDoc in Listing 1.

At the end of the entire process of retrieving, parsing and displaying the document, the OnEndRetrieval event handler is called (automatically). We can use this event handler to display the fact that we've just successfully retrieved and displayed the URL. See HTML1EndRetrieval in Listing 1.

In case an error occurs while retrieving the document, the OnError event handler is called, which we can use to display the error (bad) status for the URL. See HTML1Error in Listing 1. Of course, all kinds of errors can occur, which is why this event handler has so many parameters. Most often, however, the document simply cannot be found (BAD URL), so we won't go any deeper into the HTML error codes at this time.

Local URLs

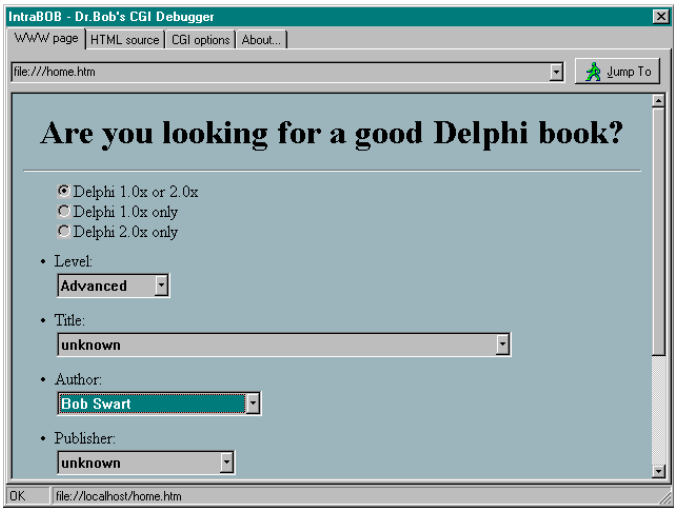
Normally, a URL consists of something that starts with http:// to show we want to retrieve a document from the internet. But what if we want to view a document from the network or even a local disk? In that case we need to start the URL with file:/// and the filename, such as file:///d:/www/home.htm. Assuming that the local file home.htm contains the HTML file with the TDMBKS on-line CGI-FORM, our home-made Web Browser would display it as shown in Figure 1.

Hardly any difference from a 'real' browser, right? Well, just wait, it gets even better (you did notice the other three tabs on the PageControl, right?). There's only one reason why I'm not using the HTML ActiveX control to write my own real Web Browser: it doesn't support HTML 3.0 (no frames), nor does it support any of the Netscape 3.0 or Explorer 3.0 extensions. Unfortunately, I don't know if the HTML ActiveX control will be upgraded to include enhanced HTML support.

HTML Forms

Now that we've played a little with the HTML ActiveX control and its events, let's take a look at its

➤ *Figure 1*




```
procedure TDebugCGI.JumpBtnClick(Sender: TObject);
begin
  StatusBar.Panels[0].Text := '@'; { progress }
  if ComboBox.Text <> '' then
    HTML1.RequestDoc(ComboBox.Text)
end;
procedure TDebugCGI.HTML1DoRequestDoc(Sender: TObject; const URL: string;
  const Element, DocInput: Variant; var EnableDefault: Wordbool);
begin
  StatusBar.Panels[0].Text := '@@'; { progress }
  StatusBar.Panels[1].Text := 'Contacting ' +URL
end;
procedure TDebugCGI.HTML1EndRetrieval(Sender: TObject);
begin
  StatusBar.Panels[0].Text := 'OK'; { success }
  StatusBar.Panels[1].Text := HTML1.URL;
end;
procedure TDebugCGI.HTML1Error(Sender: TObject; Number: Smallint;
  var Description: string; Scode: Integer; const Source, HelpFile: string;
  HelpContext: Integer; var CancelDisplay: Wordbool);
begin
  StatusBar.Panels[0].Text := 'BAD'; { error }
  StatusBar.Panels[1].Text := HTML1.URL;
end;
```

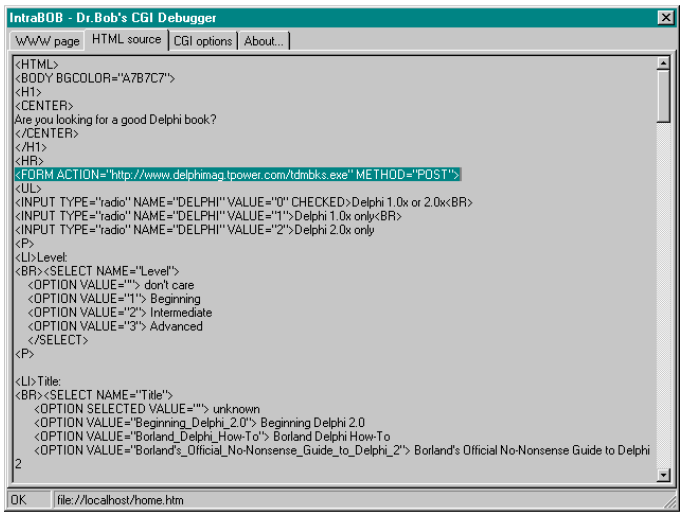
➤ *Listing 1*

properties. One seems to be especially useful for our CGI Debugger: the Form property. This is a collection of the forms which are contained in the HTML page. Unfortunately, the type of this property seems to be an array of Variants, with almost no documentation on the real contents or how to get to it. The only thing I could find out was the fact that the Forms class has a Count property for the number of Forms inside the HTML page, and an Item (default) index property to get to the individual Forms themselves. As for the Form Variant itself, it has three properties: Method (GET or POST), URL (the CGI program that's being called) and URLEncodedBody which contains the data that is sent to the Web Server.

This all sounds great in theory. However, in practice it seems that you really can't get any useful information out of the Forms property before you actually submit a form. And in order to be able to test, debug or configure a local CGI application, it would be nice to get at least some information beforehand, like the URL of the remote CGI application to execute.

The only solution I could find was to get to the source of the HTML page itself. Fortunately, the HTML ActiveX control has another useful property which does work, the SourceText property of type String which contains the entire document. Of course, this property only has a valid value right after the retrieval and parsing of the document is done. So, we can modify the OnEndRetrieval event handler to assign the SourceText to a TMemo as shown in Listing 2.

Let's assume that this Memo is placed on the second page of the PageControl, then the result of requesting the home.htm page with the



➤ *Figure 2*

```
procedure TDebugCGI.HTML1EndRetrieval(Sender: TObject);
var Str: ShortString;
    i: Integer;
begin
  StatusBar.Panels[0].Text := 'OK';
  StatusBar.Panels[1].Text := HTML1.URL;
  Source.Lines.Clear;
  Source.Lines[0] := HTML1.SourceText;
  i := 0;
  Str := '';
  repeat
    Str := UpperCase(Source.Lines[i]);
    if Pos('ACTION=',Str) > 0 then begin
      Str := Copy(Source.Lines[i],Pos('ACTION=',Str)+8,255);
      Delete(Str,Pos('"',Str),255)
    end else
      Str := '';
    Inc(i)
  until (i >= Source.Lines.Count) or (Length(Str) > 0);
  RemoteCGI.Text := Str
end;
```

➤ *Listing 2*

TDMBKS CGI Form looks like Figure 2. This clearly shows the (highlighted) line we're looking for: the `FORM ACTION` part which contains the `ACTION URL` of the remote CGI application that is to be executed. In this case, it's the TDMBKS CGI application at `http://www.delphimag.tpower.com/tdmbks.exe` and the method used is `POST`.

To get the value of the `ACTION URL`, we just have to scan the contents of the `Memo` for a line that contains the sub-string `ACTION=`. We can best do this in the `OnEndRetrieval` event handler again, right after we've set the contents of the `Source Memo`, as shown in Listing 2.

Note that we stop looking for an `ACTION URL` as soon as we've found the first one. In theory, an HTML page could contain more than one CGI Form (in fact, Steve showed us a standard CGI form and WinCGI form on the same HTML page back in Issue 16). However, in these rare cases, we should be able to split the HTML page into sub-pages, where each page contains one CGI Form only. Since multiple CGI Forms on an HTML page do not influence each other, it is not necessary for testing and debugging purposes to keep them together.

CGI Options

Now that we have obtained the remote `ACTION URL`, it's time to specify some additional CGI options. First of all, the main purpose of this CGI Debugging tool is be to be able to debug a CGI application while it's on our local disk, rather than executed on a remote web server. So, we must show the remote `ACTION URL` and then ask for the name and location of the local CGI executable. By default, we'll take the filename part from the `ACTION URL` and assume the executable will be on the `PATH` or in the current directory (the user can enter another executable or browse the disk). Once the local CGI application is known, we can execute it on our local machine, just like any other local executable, using `WinExecAndWait` (so we wait until it's done). We only need to supply it with the input it expects, and for that we need to specify the specific type of CGI application we're dealing with.

There are at least two different forms of CGI, standard CGI and a higher level called WinCGI. As we learned in Issue 15, the first format uses environment variables and the standard input and output files, and (as shown in Issue 16) the latter uses a Windows INI format file that specifies the names for the input and output files to communicate between the Web Browser and the Web Server. Delphi 2 CGI applications are typically non-visual command-line or `CONSOLE` applications, where the input contains the information (request) sent by the client and the output is the dynamic HTML document which is generated on the fly (and sent back to the Web Browser).

So, we need to know what type of CGI application we're about to debug. Since we want to set up the information to run the application locally and we can't just smell or guess what type it is, the user must specify the type. In the case of a WinCGI application we also need to specify the name of the INI file where the information is stored.

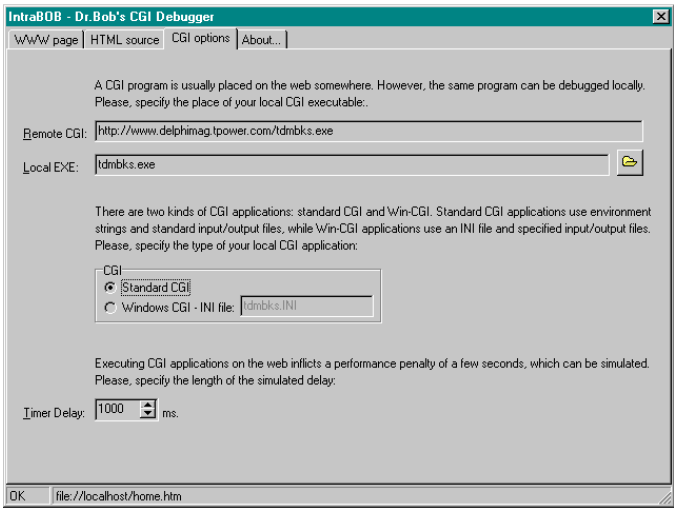
Three edit controls are used for the remote CGI executable, the local CGI executable and (if required) the INI file for the WinCGI executable. If the contents of the edit field for the remote CGI changes, the contents of the local CGI and INI file fields change as well. The same holds for changes in the local CGI application, which are reflected automatically in the name of the INI file. This is implemented by connecting the `OnChange` events of the first two edit fields to set the text of the latter two edit fields as shown in Listing 3.

`RemoteCGIChange` extracts the EXE name from the remote CGI application and assigns it to the local CGI application, while `LocalCGIChange` just changes the filename extension from the local CGI application to INI for the INI file.

This leaves us with one last CGI configuration option. In the real world, executing a CGI application (on a Web Server) costs a certain amount of time. To simulate this, we can use a `SpinButton` to set the number of milliseconds our CGI Debugger waits until the resulting form is shown again. As I found the HTML ActiveX control to be somewhat less reliable when trying to read the resulting file too soon (ie right after the `WinExecAndWait`), it's not a bad idea to wait at least a few milliseconds anyway.

All these options can be set in the CGI Options page, shown in Figure 3.

➤ *Figure 3*




```
procedure TDebugCGI.RemoteCGIChange(Sender: TObject);
var
  Str: ShortString;
  i: Integer;
begin
  Str := RemoteCGI.Text;
  i := Length(Str);
  while (i > 0) and (Str[i] <> '/') do Dec(i);
  if i > 0 then Delete(Str,1,i);
  LocalCGI.Text := Str { just the EXE-name }
end;
procedure TDebugCGI.LocalCGIChange(Sender: TObject);
begin
  IniFileName.Text := ChangeFileExt(LocalCGI.Text,'.INI')
end;
```

➤ *Listing 3*

```
procedure TDebugCGI.HTML1DoRequestSubmit(Sender: TObject; const URL: string;
  const Form, DocOutput: Variant; var EnableDefault: Wordbool);
var f: System.Text;
    Exe: String;
begin
  if RadioGroup.ItemIndex = 0 then begin
    { standard CGI }
    System.Assign(f,'data');
    Rewrite(f);
    writeln(f,Form.URLEncodedBody);
    System.Close(f);
    System.Assign(f,'cgi.bat');
    Rewrite(f);
    writeln(f,'set REQUEST_METHOD=POST');
    writeln(f,'set CONTENT_LENGTH=',Length(Form.URLEncodedBody));
    writeln(f,LocalCGI.Text,' < data > output.htm');
    System.Close(f);
    Str := 'COMMAND.COM';
    with TBDosEnvironment.Create(Self) do
    try
      Str := GetDosEnvStr('COMSPEC');
    finally
      Free
    end;
    Str := Str+' /C '+ExtractFilePath(Application.ExeName)+'cgi.bat'
  end;
  WinExecAndWait(PChar(Exe),SW_NORMAL);
  ComboBox.Text := 'file:///output.htm';
  Timer.Interval := SpinEdit.Value;
  Timer.Enabled := True
end;
```

➤ *Listing 4*

Submit

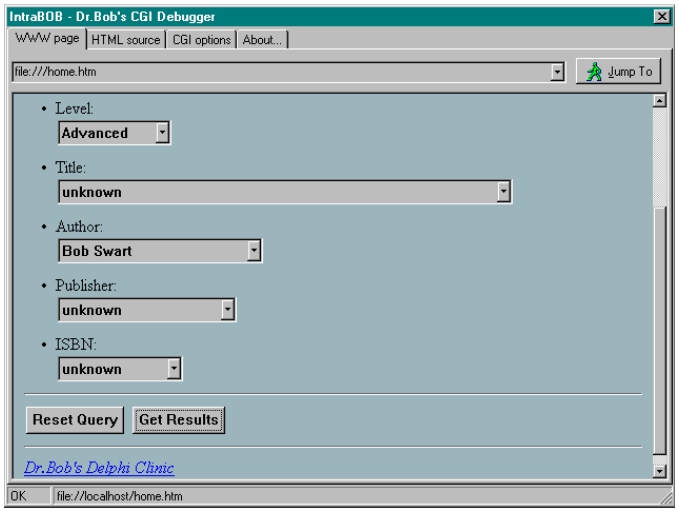
Now we're ready to do something with all the information we've collected. Only one question remains: how do we know when the user has pressed the `Submit` button on a Form in an HTML Page? For this, we need to take another look at the HTML ActiveX control in the `Events` page of the Object Inspector.

Fortunately, there's one important event left that we can make good use of: `OnDoRequestSubmit`. This event is also the exact moment where the `HTML.Forms` property gets its value (which is why we couldn't use it before). The `OnDoRequestSubmit` event handler gets the `ACTION URL` as an argument and the `URLEncodedBody` in the `Form Variant` argument. For a standard CGI application (where `RadioGroup.ItemIndex` has a value of zero), we just need to dump the `Form.URLEncodedBody` data into a file (called `data`), and set two environment variables (`REQUEST_METHOD` and `CONTENT_LENGTH`) before we can execute the local CGI application. One way to do that is to generate a batch file that first sets the environment variables and then calls the executable with the data file as standard input, redirecting the output to a file called output.htm. See Listing 4. *[Shock, horror: Dr.Bob doing a quick and dirty DOS hack?! Editor]*.

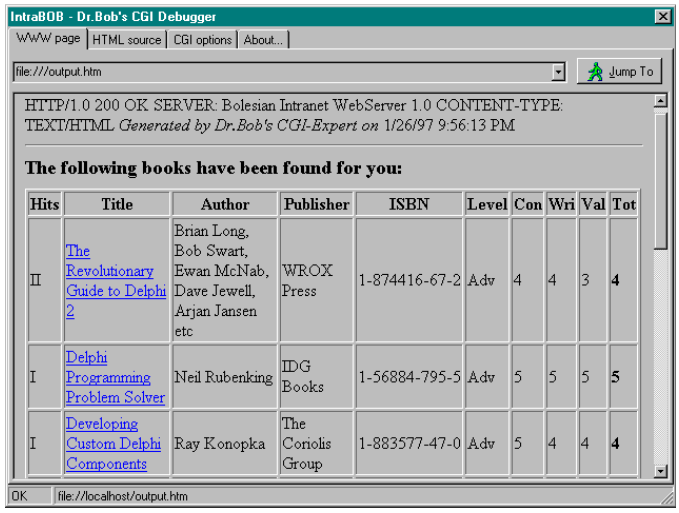
This approach makes sure no environment variable values are left when the CGI application returns. It assumes that the environment variable `COMSPEC` is defined and uses that to obtain the location of COMMAND.COM. The `/C` argument makes sure that COMMAND.COM just executes the specified cgi.bat command and then returns.

Action!

It's time to test our first local standard CGI application, TDMBKS.EXE, on a form that has been filled with some information (Figure 4). Clicking on the `Get Result` button would normally send the information to the Web Server, which in its turn will start the TDMBKS.EXE application (from the Form's action information) with the information that was filled in in this form. In this



➤ *Figure 4*



➤ *Figure 5*

case that would be:

```
DELPHI="2"
LEVEL="3"
TITLE=""
AUTHOR="Bob_Swart"
PUBLISHER=""
ISBN=""
```

Note that spaces are replaced by underscores.

Using our CGI Debugger, however, the `OnDoRequestSubmit` event handler is fired, and our code is executed, which results in the following one-line data file:

```
DELPHI=2&Level=3&Title=
  &Author=Bob_Swart
  &Publisher=&ISBN=
```

and a corresponding three-line CGI.BAT file:

```
set REQUEST_METHOD=POST
set CONTENT_LENGTH=57
tdmbks.exe < data > output.htm
```

The local TDMBKS.EXE local CGI application is able to process the passed information, perform the query and generate a dynamic HTML page on its standard output. Normally, the Web Server will then pass this dynamically generated HTML page back to the Web Browser which will show it as the resulting page to the client. In our Debugger, however, we know that the resulting output file is called OUTPUT.HTM, so when the CGI.BAT file has ended execution (we used `WinExecAndWait`), we only have to retrieve and display the local file OUTPUT.HTM, or in URL terms `file:///output.htm`, which is as shown in Figure 5.

As you can see at the top, the CGI application generates some special information (including `CONTENT-TYPE`) which is normally needed to inform the web server and browser what kind of file is received. The HTML ActiveX control just ignores this information and displays it as plain text. It doesn't matter as long as we can see what we've just generated. We can re-load the original local document at file:///home.htm and try the CGI application again and again. And we can make modifications in the CGI application and test it again. All without having to upload it to the web server. We can even test CGI applications on a portable PC with this tool (so you can give demonstrations showing sample CGI applications without having to set up a real internet connection, LAN intranet or personal web server).

Testing WinCGI

For a WinCGI executable, we need to perform other actions inside the `DoRequestSubmit` event handler (remember that the checked property of the CGI-type `RadioGroup` will tell us which type of CGI executable we've selected). For a WinCGI executable, we need to create an INI file with three sections: one for `[cgi]`, one for `[System]` and one for `[Form Literal]`. The first one contains the length of the `URLEncodedBody` string, the second contains the name of the output file (the OUTPUT.HTM file where the resulting HTML page needs to be written), while the third contains the `URLEncodedBody`, split up into the individual fields with their values. See Listing 5.

This time, the command to execute simply consists of the local CGI application followed by the name of the INI file as a command line argument. This way, the WinCGI application can read the INI file and simply obtain the required information from it (as Steve showed back in Issue 16).

As we saw earlier in the standard CGI data file, the `URLEncodedBody` consists of each field, followed by an `=`, followed by the value of the field, separated by an `&` sign. This means that we just have to split the `URLEncodedBody` into sub-parts that are separated by `&` and write the sub-parts as simple strings (they are already in INI file format for us, which saves us some additional work). The resulting INI file for the same query we saw with the standard CGI example in Listing 6.

Normally, a lot more information is passed in the INI file for a WinCGI application. However, I personally have found that most of this information is just generated and passed and not really used by (or useful for) the WinCGI application, so instead of trying to mimic a WinCGI web server, let's just make sure we can test our applications.

Enhancements

Now that we can configure and test standard CGI and WinCGI applications, what's next? Well, we could include a CGI Data page on the PageControl, listing the same information that is generated for the standard CGI or WinCGI applications, and maybe a special Debug Info page where we could list and trace each step (like looking up a new URL). We could even include an option to view and modify the CGI data before it's actually sent to the local CGI application, so we can test and monitor the behaviour of our CGI applications exactly.

A few of these ideas, along with some other enhancements, will be implemented in the next version of this *free* utility, which I have named IntraBOB (Hey, it's for free, so I can pick the name, right?!). Check my home page, as by the time you read this it should be available.

Conclusion

I hope to have shown that we can not only write, but also configure and test, CGI applications using Delphi 2 in a fairly straightforward manner. Full source code is available on this month's cover disk, including the source code for `WinExecAndWait`, `Delay` and the `TBDosEnvironment` component used in this utility. If you have any experience, trouble or interesting ideas about internet development using Delphi, don't hesitate to send me a message at bob@bolesian.nl.

Bob Swart (home.pi.net/~drbob/) is a professional knowledge engineer technical consultant using Delphi and C++ for Bolesian (www.bolesian.com), free-lance technical author for The Delphi Magazine, and co-author of *The Revolutionary Guide to Delphi 2*. Bob is now co-working on *Delphi Internet Solutions*, a new book about Delphi and the internet. In his spare time, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 3-year old son Erik Mark Pascal and his 4-month old daughter Natasha Louise Delphine.

```
procedure TDebugCGI.HTML1DoRequestSubmit(Sender: TObject; const URL:
  string; const Form, DocOutput: Variant; var EnableDefault: Wordbool);
var
  f: System.Text;
  Str: String;
  i: Integer;
begin
  if RadioGroup.ItemIndex = 0 then begin
    { WinCGI }
    System.Assign(f,IniFileName.Text);
    Rewrite(f);
    writeln(f,'[cgi]');
    writeln(f,'URL=',URL);
    writeln(f,'Content Length=',Length(Form.URLEncodedBody));
    writeln(f);
    writeln(f,'[System]');
    writeln(f,'Output File=output.htm');
    writeln(f);
    writeln(f,'[Form Literal]');
    Str := Form.URLEncodedBody;
    i := Pos('&',Str);
    while (i > 0) do begin
      writeln(f,Copy(Str,1,i-1));
      Delete(Str,1,i);
      i := Pos('&',Str)
    end;
    System.Close(f);
    Str := LocalCGI.Text + ' ' + IniFileName.Text
  end;
  WinExecAndWait(PChar(Str),SW_NORMAL);
  ComboBox.Text := 'file:///output.htm';
  Timer.Interval := SpinEdit.Value;
  Timer.Enabled := True
end;
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
[cgi]
URL=http://www.delphimag.tpower.com:80/tdmbks.exe
Content Length=57

[System]
Output File=output.htm

[Form Literal]
DELPHI=2
Level=3
Title=
Author=Bob_Swart
Publisher=
```