Under Construction: Delphi 5 WebBroker Stuff

by Bob Swart

Over the past few months, we have examined Delphi 5 InternetExpress, the extension of WebBroker and MIDAS. And although I can hardly get enough of InternetExpress and XML, we must not forget the WebBroker technology itself, which was also enhanced in Delphi 5. So let's take a closer look. And for those die-hard InternetExpress fans: we'll take our first steps at creating custom InternetExpress components (and we'll leave XML at home this time).

WebBroker

There are a number of changes in the WebBroker components and wizards. Most of them are bug fixes or workarounds for existing problems, but some are quite handy. We'll examine a number of them while implementing a new internet application that I currently use on my website. It's about banner advertising: the ability to rotate banners on a specific location, measuring the number of impressions as well as the click-through rate (when someone actually clicks on the banner and jumps to the advertiser's website). This functionality is implemented in a single Delphi 5 WebBroker application, with the measured results available using a bit of help from the InternetExpress components (this time truly without the need for a runtime licence fee!).

Runtime Packages

One change to Web Modules which has a lot of impact is the fact that we can now use runtime packages when building WebBroker ISAPI or NSAPI DLLs. In previous versions of Delphi, this was not possible (and the result was often really big DLLs). Now, we can compile these ISAPI DLLs using runtime packages, which makes them much smaller, therefore easier to distribute and update (of course, we still need to distribute the packages as well, but at least these can be reused by other applications on the web server).

In order to facilitate this change, a number of things were taken from the `HTTPApp` unit and moved to a new unit called `WebBroker`. This change affects all existing WebBroker projects written in previous versions of Delphi as, from now on, all WebBroker applications must have the `WebBroker` unit in their `uses` clause. This unit holds the definition of the global WebBroker `Application` variable. In order to 'upgrade' existing WebBroker applications to Delphi 5, we must insert the `WebBroker` unit into the `uses` clause of the project file (and optionally remove the `HTTPApp` unit).This *should* automatically be the case with all WebBroker applications that we create with Delphi 5. I say *should*, because unfortunately, this is not the case.

The *DB Web Application Wizard*, not updated since Delphi 4, does not generate code that includes the `WebBroker` unit. The Delphi IDE probably detects that it uses an `Application` object, without an `Application` object in scope, so it duly adds a missing unit... the `Forms` unit. And yes, this project compiles just fine. And it even runs, although it doesn't do much. You won't even get an error message, you just get nothing. The obvious fix is to add the `WebBroker` unit (since you don't need the `Forms` unit you can remove the `Forms` reference altogether).

Talking about the *DB Web Application Wizard*, it still only generates a single table-based or query-based `DataSetTableProducer`, so I still find it a bit taxing to call this a true 'wizard' in what it does. There's another nuisance I only recently stumbled upon. I find it convenient to add a new project in the Project Manager, instead of starting a separate new project. However, when you want to add a new project from the Project Manager, you'll find that the `Business` tab (which contains the *DB Web Application Wizard* icon) is not available. And hence the *DB Web Application Wizard* is not available either. Obviously, I find this not a big problem (since that wizard is buggy to begin with), but I have a feeling this was unintended (and untested perhaps) as well.

Rotating Banners

Let's skip the bugs for now (we'll encounter some more in a moment), and start our bitmap banner rotating web server application: BoBanner. As I explained last time, an ISAPI DLL is easy to debug, while a CGI application is easy to deploy (and update), so I always create both an ISAPI DLL project and a CGI EXE project and let them share the same web module unit. It works really well, especially since we have a Project Manager to hold both projects at the same time.

Once we have both projects, we can open up the web module unit. The first thing that's new is that a web module is derived from a regular data module, and as a consequence we also have the Visual Data Module Designer available for web modules. This is handy when working with tables, but we won't be using any tables today. Nevertheless, the Visual Data Module Designer is a big help, and I wonder what enhancements will be made to it in C++Builder 5 and Delphi 6 (one can always hope for even more, right?).

Action Enhancements

Right click on the web module to pop up the Action Editor. Right

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var FileStream: TFileStream;
begin
  try
    FileStream := TFileStream.Create('d:\www\drbob42\gif\javahelp.gif',
      fmOpenRead OR fmShareDenyNone);
    try
      FileStream.Position := 0;
      Response.ContentType := 'image/gif';
      Response.ContentStream := FileStream;
      Response.SendResponse; // send header + Stream
    finally
      FileStream.Free
    end
  except
  end
end;
```

➤ *Listing 1: Returning a single bitmap banner.*

click again to add two new web action items. If you select one of them, you'll notice a new property called `Producer`. This can be used to specify any `PageProducer` or `TableProducer` that's available on the web module, and replaces the code we used to write in the `OnAction` event handler. So, an `OnAction` event handler that does the following:

```
Response.Content :=
  PageProducer.Content;
```

actually does the same thing as assigning `MyPageProducer` to the `Producer` property of the `WebActionItem`. The latter uses no code, which may be convenient, but also removes the ability to set a breakpoint in that line of code. Also, since I usually produce more than the output from a single `Producer`, I find myself using the `Producer` property very little. But it can be handy sometimes.

Non-HTML Producing

In previous articles on WebBroker (or InternetExpress) I've mainly concerned myself with the production of HTML: dynamic web pages. This time, however, the objective is to produce a random rotating bitmap banner which is displayed on top of a static web page. And that's a bit different, since we can no longer simply assign something to the `Response.Content` string (a binary bitmap may contain `#0` characters and so cannot be put in a `String`, as this could inadvertently signal the end of the `PChar` inside the `String`).

Whenever we need to return something other than a simple `text/html`, we first need to specify the correct `ContentType` (which is set to `text/html` by default). In our case, we need to set it to `image/gif`, but there are numerous other mime options available (including audio or video streaming). After we've set the correct content type, so the browser knows how to interpret the data that follows, it's time to send the binary content itself. Not by using the `Response.Content` string, but by streaming it out using the `ContentStream` property and the `SendResponse` method. If we need to stream more binary stuff, we need subsequent calls to the `SendStream` method (which does not send the header information again).

In order to return an `image/gif` bitmap banner stored in the file located at d:\www\drbob42\gif\javahelp.gif (accessible from the web server, of course), we can write the code in Listing 1.

We need to open the file using the `fmOpenRead OR fmShareDenyNone` options since we only need to read it, and we don't want to stop other applications from accessing the same file. Note that we need to create our own instance of a `FileStream` component, assign it to the `Response.ContentStream` property, send it to the client (using the `SendResponse` method) and only then can we free the `FileStream` again.

The HTML fragment to call the above script and produce the dynamic bitmap banner is as follows (without the line breaks):

```
SRC="http://www.drbob42.com/
  cgi-bin/bobanner.exe"
  BORDER=0
```

Note the `BORDER=0` part, which will come in handy in a moment, since these bitmap banners are always clickable, meaning that a hyperlink is defined to take you elsewhere when you click on it.

Rotating Banners

In order to show a random banner from a list of banners, thereby giving the illusion of 'rotating' banners (a different one for each visitor), we simply need to prepare a list of bitmap banner locations, and make a call to the `Random` function to pick one. Here's a tip: if you put this in a CGI executable, make sure you call the `Randomize` function somewhere in your application's or unit's `initialization` section, otherwise you will always return exactly the same bitmap banner. The code can be seen in Listing 2.

➤ *Listing 2: Returning a random bitmap banner.*

```
procedure TWebModule1.WebModule1WebActionItem2Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
const
  MaxBanners = 4;
  Banners: Array[0..MaxBanners-1] of String =
    ('d:\www\drbob42\gif\robohlp1.gif', 'd:\www\drbob42\gif\robohlp2.gif',
     'd:\www\drbob42\gif\robohlp3.gif', 'd:\www\drbob42\gif\robohlp4.gif');
var FileStream: TFileStream;
begin
  try
    FileStream := TFileStream.Create(
      Banners[Random(MaxBanners)],
      fmOpenRead OR fmShareDenyNone);
    try
      FileStream.Position := 0;
      Response.ContentType := 'image/gif';
      Response.ContentStream := FileStream;
      Response.SendResponse; // send header + Stream
    finally
      FileStream.Free
    end
  except
  end
end;
```

Note that both Listings contained an empty except...end clause. An exception can be raised if the bitmap banner files cannot be found on the web server, in which case the normal 'document contains no data' exception is raised (since we didn't assign anything to the Content variable itself).

Redirections, Anyone?

For each time we call a SendResponse, we should also update a logfile with the current date/time and the specific bitmap which is shown. This results in an accurate list of impressions. What's even more useful for advertisers, however, is a list of people clicking on the bitmap banner, also called the click-through rate. Usually, when people click on a bitmap banner, they are sent to the website of the advertiser (sometimes to a special entrance page, sometimes directly to the homepage or a specific product page). This redirection can also be done using web modules, as well as logging the clicks themselves.

Assuming we create a new WebItemAction with the /redirect PathInfo, we can write the single line of code as in Listing 3 to handle the redirection.

The HTML fragment in Listing 4 (on a single line) can be used for the hyperlink as well as the dynamic bitmap banner.

Again, I didn't include code for logging the click-through (nor did I include this code when showing the bitmap banner in the first place), but that's usually pretty trivial anyway, and is in the code on this month's disk.

InterBuilder Express?

Anyone remember IntraBuilder? The hype with which it was announced at a Borland Conference back in, I believe, 1996 was only surpassed by the silence with which it was killed off a year or so ago. At that time, people could freely exchange their IntraBuilder box for their choice of JBuilder or Delphi with WebBroker. I'd pick the latter one, of course, although WebBroker never did contain the easy way of visually crafting HTML web pages like IntraBuilder did back in those days.

And now we have InternetExpress as an additional feature set of WebBroker. In the past few months, we've examined the multi-tier and XML extensions in depth, but we haven't actually grasped the true power of the Web Page Editor that comes with InternetExpress. And that's a shame, because it finally offers us some (limited) WYSIWYG web page design functionality. Let's go back to our bitmap banner advertising example and see how we can open up and generate reports of the logfiles, but for the specific advertisers and webmaster only, of course. For that, we need some kind of login dialog, and that's where I'll be using InternetExpress, without XML.

InternetExpress: No XML

Drop a MidasPageProducer component on the web module. Don't even bother with an XMLBroker or any of the Connection components, because we are not using any data-aware stuff anyway. We only need the MidasPageProducer to be able to start the Web Page Editor (wouldn't it be nice to have a PageProducer with a more neutral name than Midas...?). I'm not even using MIDAS and I certainly don't expect to be liable for MIDAS runtime licence fees.

Anyway, once we have dropped a MidasPageProducer on the web module, right click on it to start the Web Page Editor. Last time, we started with a DataForm, but this time we need a QueryForm to implement a login dialog. So, right click on the upper-left pane of the Web Page Editor and select New Component (or click the Ins button) to get the Add Web Component dialog and select the QueryForm component.

A QueryForm can be used to fire another action (typically from the same web module), and that's what the Action property is all about. Unfortunately, I'd much rather work with a specific PathInfo property, since now I need to fill in the exact entire action (including the path of my web server application itself, which depends on the location on my development/debug machine and the deployment web server). For now, I just specify:

```
http://192.168.91.201/cgi-bin/
  bobanner.exe
```

as the Action property, which points to my local development machine. Note that I should also specify the PathInfo here, so that becomes something like:

```
http://192.168.91.201/cgi-bin/
  bobanner.exe/admin
```

Which means we need to implement another WebItemAction to implement the /admin pathinfo. The dialog itself, built on top of the QueryForm, should contain a listbox of possible advertiser names and an editbox with the password. We should also have Login and Cancel or Close buttons. Right click on the QueryForm, and select Add New Component again to add both a QueryFieldGroup and the QueryButtons. The QueryButtons by default contain the Submit and Reset buttons. We'll get back to those in a minute. First, make sure the QueryFieldGroup (which is currently empty) is positioned higher than the QueryButtons. Now, select and right click on the QueryFieldGroup to see the list of relevant QueryForm subcomponents, which is quite long, and contains, among others, both

```
procedure TWebModule1.WebModule1WebActionItem3Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.SendRedirect(Request.QueryFields.Values['URL'])
end;
```

➤ *Above: Listing 3*

➤ *Below: Listing 4*

```
<A HREF="http://www.drbob42.com/cgi-bin/bobanner.exe/redirect?
  URL=http://www.domain.com"> <IMG SRC="http://www.drbob42.com/cgi-bin/
  bobanner.exe" BORDER=0></A>
```

a QuerySelectOptions and a QueryPassword component. The former can be used to select from a predefined (or data-bound) list of items, while the latter can be used to allow the end-user to enter a password without the characters being readable at runtime (ie you only get * characters).

QuerySelectOptions

Select the QuerySelectOptions component, which is a powerful little gem. The easiest way to use this component is to click on the ellipsis next to the Items property and enter a number of items (like five email addresses that can be used as login names). The HTML tab of the Web Page Editor shows the corresponding HTML which is being generated, while the Browser tab enables us to click on the combobox to see the items we've just entered. And if you don't like a combobox, you can change the DisplayRows property from -1 to 3 to get a listbox instead.

Another way to use a QuerySelectOptions component is to use the DataSet, FieldName, ValuesField and ItemsField properties. The DataSet and FieldName obviously connect to the dataset and field whose value is to be set, while the ValuesField contains the values we set (like a User ID), and the ItemsField contains the values we see in the browser (like the full name or email address of the user). Quite powerful and, even in its data-aware edition, not dependent on XML or any MIDAS technology whatsoever (you can easily connect to a local table, in fact, you actually need a real DataSet and not an XMLBroker component to connect to in the first place). And the best thing is we can actually see and manipulate the data and the web page at design-time. The Web Page Editor uses the Internet Explorer ActiveX control as its internal browser, so this is a true WYSIWYG interface.

Check, Double Check?

Apart from a QueryPassword and QuerySelectOptions component, there are many others available to use with a QueryForm. These include a QueryText (single edit field), QueryTextArea (memo field), QueryRadioGroup (set of radio buttons) and even QueryHiddenText (to handle state information, for example). One of the things I'm missing, which is usually available when building CGI Forms in HTML, is a checkbox, or rather a QueryCheckbox component. I have no idea why it's not here, but I need one right now. I'd like to have an option called 'Email Report', which just emails the results (hence the email addresses as user name) instead of producing an HTML report.

But how do we create a QueryCheckbox component? And no, I don't want to use a QueryRadioGroup with the 'Email Report' and 'Report Online' options, nor do I want a dropdown combobox with these options. I just want to use a single checkbox, unchecked by default, which says 'Email Report'. That's it. And that's also the cue for another step deeper inside InternetExpress, since it looks like we need to write an InternetExpress custom component.

```
function TWebCheckbox.ControlContent(Options: TWebContentOptions): string;
var Attrs: string;
begin
  AddAttributes(Attrs);
  Result := Format('<INPUT TYPE=CHECKBOX %0:s>', [Attrs]);
end;
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
TQueryCheckbox = class(TWebCheckbox, IQueryField)
private
  FText: string;
protected
  function GetText: string;
  procedure SetText(const Value: string);
public
  class function IsQueryField: Boolean; override;
end;
```

InternetExpress Components

The best way to find out how InternetExpress components are built, is to take a look at the InternetExpress Sample Components package (which contains the QueryPassword component, by the way, so you need to install this package if you need to recreate the code for this month).

These components can be found in the directory DEMOS\MIDAS\INTERNETEXPRESS\INETXCUSTOM. It appears that a lot of the InternetExpress components are not simply components in a VCL hierarchy, but also contain (and implement) several different interfaces, like IQueryField, which is required for a component on a QueryForm.

One of the most simple base classes is the TWebTextInput component, which we can use to derive a simple TWebCheckBox component from. There's one core function, called ControlContent, which returns the HTML fragment for that particular component. This is the core method that we need to override for all custom InternetExpress components, along with some other methods and interface methods should need arise. For now, let's implement the ControlContent method to return a CheckBox instead of a simple edit textbox, see Listing 5.

This small method is almost entirely copied from the TWebTextInput component (I left out some JavaScript event handling code), where instead of an <INPUT TYPE=TEXT %0:s> we specify a TYPE=CHECKBOX to get a checkbox instead of an editbox. It couldn't be simpler, and frankly I'm amazed this component isn't in Delphi 5 Enterprise in the first place!

Apart from overriding the ControlContent method, we need to publish a few properties (which are not published in the TWebTextInput component, yet), see Listing 7 for more details. Once you've added this component to a Delphi 5 package, we can use the TWebCheckbox component, but only in regular DataForms. Not in QueryForms, yet.

QueryCheckbox

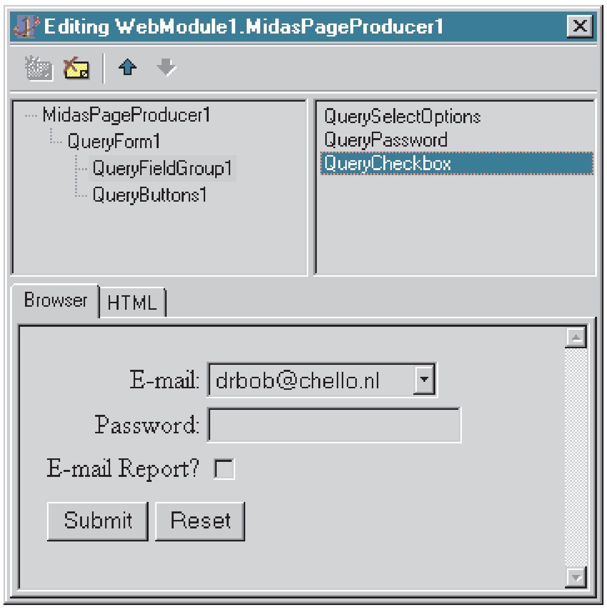
The main reason I started with a regular TWebCheckbox component, while I needed a TQueryCheckbox component, is that it's fairly easy to turn a 'regular' component into a 'query' component, once you know how to do it. In fact, the only thing to do is derive from the 'regular' class, combined with the IQueryField interface class, and implement the necessary IQueryField interface methods, see Listing 6.

The class function IsQueryField is the most important. Class function means that it can be called on the class type TQueryCheckbox itself, without the need for an instance (it also means that from within that method we can't get to an instance or instance data, but that's another story). The class function IsQueryField should return True.

Which leaves the GetText and SetText methods, connected to the FText private data placeholder. I'm sure the implementation of these one-liners won't surprise you (Listing 8), and at this time I can only tell you that these methods are needed for the IQueryField interface, so we need to implement them no matter what (we'll get back to this topic at some other time, promise!).

After implementing the three methods for the TQueryCheckbox component, we can add that component to the package, and are

➤ *Figure 1*



E-mail: drbob@chello.nl

ready to add a real QueryCheckbox component to our QueryForm on the MidasPageProducer.

The name of the checkbox will be the name of the variable that's passed to the web server application (with the value 'on' if checked, and 'off' if not checked). Quite useful, as you can imagine, and now available for Delphi 5 InternetExpress users (and readers of this column).

The full implementation of the TWebCheckbox and TQueryCheckbox is shown in Listing 7.

Apart from the default WebActionItem (which shows a fixed bitmap banner) and the WebActionItem with PathInfo set to /banner (which shows a random banner from a list of four) and /redirect (which jumps to another URL), we now need to implement the PathInfo's /login (which shows the above MidasPageProducer) and /admin (which shows the result when logging in).

Right click on the web module and create two new WebItemActions (one for /login and one for /admin). Select the /login action. Now we can use the Producer property: assign it to MidasPageProducer. When this action is 'executed', the MidasPageProducer produces the output for it. Like I said before, this eliminates the need to write an OnAction event handler for this action. If an OnAction event handler is present and the Producer property is assigned, the Producer property is used to assign the Response.Content, followed by the call to the OnAction event handler (which will have a Response argument with a Content property already filled in). So it's not a case of one-or-the-other, but actually one after the other.

PageProducing

The /admin WebActionItem will use a regular PageProducer to produce its

➤ *Listing 7*

```
unit DrBob42X;
interface
uses
  Classes, HTTPApp, WebComp, MidItems;
type
  TWebCheckbox = class(TWebTextInput)
  protected
    function ControlContent(Options: TWebContentOptions):
      string; override;
  published
    property DisplayWidth;
    property ReadOnly;
    property Caption;
    property CaptionAttributes;
    property CaptionPosition;
    property TabIndex;
    property Style;
    property Custom;
    property StyleRule;
  end;
  TQueryCheckbox = class(TWebCheckbox, IQueryField)
  private
    FText: string;
  protected
    function GetText: string;
    procedure SetText(const Value: string);
  public
    class function IsQueryField: Boolean; override;
  end;
procedure Register;
implementation
uses
  SysUtils;
{ TWebCheckbox }
function TWebCheckbox.ControlContent(
  Options: TWebContentOptions): string;
var Attrs: string;
begin
  AddAttributes(Attrs);
  Result := Format('<INPUT TYPE=CHECKBOX %0:s>', [Attrs]);
end;
{ TQueryCheckbox }
class function TQueryCheckbox.IsQueryField: Boolean;
begin
  Result := True;
end;
function TQueryCheckbox.GetText: string;
begin
  Result := FText;
end;
procedure TQueryCheckbox.SetText(const Value: string);
begin
  FText := Value;
end;
{ Register procedure }
procedure Register;
begin
  RegisterWebComponents([TWebCheckBox,TQueryCheckbox]);
end;
end.
```

dynamic HTML. And this component, together with its cousin DataSetPageProducer, has been extended with an additional property as well: the StripParamQuotes property. This property is a blessing for those of us that used HTML editors like FrontPage (instead of the more user-friendly Notepad, which is my first choice). As you may remember, the PageProducer components make use of special HTML #-tags in their HTMLDoc or HTMLFile specified property. And these #-tags can also contain special parameters, in the form of: TAG Param=Value. Some web editors don't recognise these parameters, and will embed them in quotation marks, which the OnHTMLTag event of the PageProducers does not like. As of Delphi 5, we can set StripParamQuotes to True to indicate that the PageProducer should remove the quotes around the parameters. So, from now on it's safe to use FrontPage again, although it still produces almost unreadable HTML...

Obviously, we can connect the /admin PathInfo to the PageProducer that produces the HTML report.

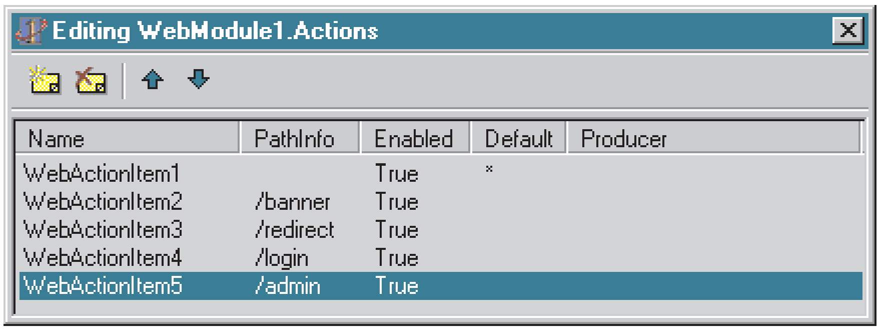
The final code, with the AddLogEntry and LogEntries support methods, is shown in Listing 8.

Note that the AddLogEntry and LogEntries methods use a common logfile. More important, is the fact that AddLogEntry only 'appends' to this logfile, so you have to be sure that an empty logfile already exists (to append to), otherwise you may need to perform a 'rewrite' the first time (when the appends fails). Since the rewrite is only needed for a new logfile, like a monthly edition, I've left it out of the listing.

The final banner-showing, redirection and administration WebBroker app is operational on my website at www. drbob42.com (do take a look!). With a little help from InternetExpress as 'WebBroker++'.

➤ *Listing 8*

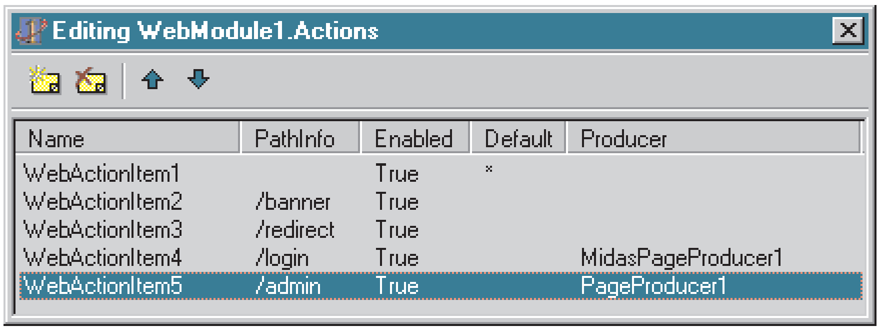
```
unit WebMod;
interface
uses
  Windows, Messages, SysUtils, Classes, HTTPApp,
  MidItems, QueryComps, CompProd, PagItems, MidProd,
  DrBob42X; // custom TWebCheckBox, TQueryCheckBox components
type
  TWebModule1 = class(TWebModule)
    MidasPageProducer1: TMidasPageProducer;
    QueryForm1: TQueryForm;
    QueryButtons1: TQueryButtons;
    QueryFieldGroup1: TQueryFieldGroup;
    QuerySelectOptions: TQuerySelectOptions;
    QueryPassword: TQueryPassword;
    QueryCheckbox: TQueryCheckbox;
    PageProducer1: TPageProducer;
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse;
      var Handled: Boolean);
    procedure WebModule1WebActionItem2Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse;
      var Handled: Boolean);
    procedure WebModule1WebActionItem3Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse;
      var Handled: Boolean);
    procedure PageProducer1HTMLTag(Sender: TObject; Tag:
      TTag; const TagString: String; TagParams: TStrings;
      var ReplaceText: String);
  private
    procedure AddLogEntry(Message: String);
    function LogEntries: String;
  public
  end;
var
  WebModule1: TWebModule1;
implementation
{$R *.DFM}
const
  LogFile = 'c:\logfile.txt';
procedure TWebModule1.AddLogEntry(Message: String);
var f: System.Text;
begin
  System.Assign(f,LogFile);
  Append(f);
  // start with empty file, in order to be able to append
  writeln(f,Message);
  Close(f)
end;
function TWebModule1.LogEntries: String;
var
  f: System.Text;
  Str: String;
begin
  Result := '<PRE>';
  System.Assign(f,LogFile);
  Reset(f);
  while not eof(f) do begin
    readln(f,Str);
    Result := Result + Str
  end;
  Close(f);
  Result := Result + '</PRE>'
end;
procedure TWebModule1.WebModule1WebActionItem1Action(Sender:
  TObject; Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
var FileStream: TFileStream;
begin
  try
    FileStream := TFileStream.Create(
      'd:\www\drbob42\gif\javahelp.gif',
      fmOpenRead OR fmShareDenyNone);
    try
      FileStream.Position := 0;
      Response.ContentType := 'image/gif';
      Response.ContentStream := FileStream;
      Response.SendResponse; // send header + Stream
      AddLogEntry('d:\www\drbob42\gif\javahelp.gif shown')
    finally
      FileStream.Free
    end
  except
  end
end;
procedure TWebModule1.WebModule1WebActionItem2Action(Sender:
  TObject; Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
const
  MaxBanners = 4;
  Banners: Array[0..MaxBanners-1] of String =
    ('d:\www\drbob42\gif\robohlp1.gif',
     'd:\www\drbob42\gif\robohlp2.gif',
     'd:\www\drbob42\gif\robohlp3.gif',
     'd:\www\drbob42\gif\robohlp4.gif');
var
  FileStream: TFileStream;
  Banner: Integer;
begin
  try
    Banner := Random(MaxBanners);
    FileStream := TFileStream.Create(
      Banners[Banner], fmOpenRead OR fmShareDenyNone);
    try
      FileStream.Position := 0;
      Response.ContentType := 'image/gif';
      Response.ContentStream := FileStream;
      Response.SendResponse; // send header + Stream
      AddLogEntry(Banners[Banner]+' shown (random)')
    finally
      FileStream.Free
    end
  except
  end
end;
procedure TWebModule1.WebModule1WebActionItem3Action(Sender:
  TObject; Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
var URL: String;
begin
  URL := Request.QueryFields.Values['URL'];
  Response.SendRedirect(URL);
  AddLogEntry('Redirection to '+URL)
end;
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject;
  Tag: TTag; const TagString: String; TagParams: TStrings;
  var ReplaceText: String);
begin
  if TagString = 'REPORT' then
    ReplaceText := LogEntries
end;
initialization
  Randomize;
end.
```



Editing WebModule1.Actions

Name	PathInfo	Enabled	Default	Producer
WebActionItem1		True	*	
WebActionItem2	/banner	True		
WebActionItem3	/redirect	True		
WebActionItem4	/login	True		
WebActionItem5	/admin	True		

➤ *Above: Figure 2* ➤ *Below: Figure 3*



Editing WebModule1.Actions

Name	PathInfo	Enabled	Default	Producer
WebActionItem1		True	*	
WebActionItem2	/banner	True		
WebActionItem3	/redirect	True		
WebActionItem4	/login	True		MidasPageProducer1
WebActionItem5	/admin	True		PageProducer1

Final Buggy

A final new bug that I found in the Delphi 5 WebBroker Technology is related to the WebDispatcher component. This is the component that you need to turn a regular data module into a web module (one that can Dispatch WebItemActions). Only one is allowed per data module, which is why we get an error message if we try to drop one on a web module. Unfortunately, in Delphi 5 you can drop more than one WebDispatcher component on a single data module, resulting in undesired behaviour. Of course, nobody should ever try to add a second WebDispatcher in the first place, but you can't deny it's a bug either...

Next Time

Next time in the one and only real *The Delphi Magazine*, we examine Delphi 5 CORBA stuff. At the time Delphi 5 shipped, there seemed to be little new CORBA stuff included. This may still be the case, but then again, maybe not. All the more reason to *stay tuned...* until next month.

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is an IT Consultant for TAS Advanced Technologies and a freelance technical author.