How to customise the TWebBrowser user interface (part 5 of 6)

A sample application

Exercising the code – a sample application

One of the problems that motivated this exploration was my need to use a *TWebBrowser* to display some HTML in a dialog box. The HTML needed to look and behave as if it was part of the dialog box, which could be using themes. The dialog box would also need to display a custom pop-up menu.

This sample application will emulate such a dialog box. It will use our custom *TWBContainer* to take control over the appearance of a *TWebBrowser* control. This will not be production code – for example we load the HTML from a file and we wouldn't do that in released code – but it should suffice to demonstrate that the customization class works.

The application will be developed in two stages. Stage one will be the bare application with no *TWebBrowser* customization code. We will run the program to check what it looks like in its natural state before we intervene. In stage two we'll apply the classes we've developed here and see the difference.

Stage 1

Open a new Delphi project and select the main form, then:

- Give the form a *BorderStyle* of *bsDialog*.
- Place a *TButton* centred at the bottom of the form, set its *Caption* to 'Close' and create an *OnClick* event handler for it containing just the code shown in *Listing 18*:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;  // close the application
end;
```

Listing 18

- Add a *TPopupMenu* and give it a single menu item with *Caption* set to 'Show the CSS'.
- Drop a *TWebBrowser* control, set its *Align* property to *alTop* and size it to take up most of the form. Leave room for the 'Close' button below it. Set its *PopupMenu* property to *PopupMenu1*.
- Add the `XPMan` unit (Delphi 7 and later) to the form's *uses* clause to ensure the application displays themes if available. **Note:** This may not be necessary with later Delphis that can enable themes by default in the project resource file.

Now, double-click the form to open a new *OnCreate* event handler and enter the code shown in *Listing 19*. This loads the HTML file into the browser:

```
procedure TForm1.FormCreate(Sender: TObject);
begin;
  // load content
  WebBrowser1.Navigate(
    ExtractFilePath(ParamStr(0)) + 'DlgContent.html'
  );
end;
```

Listing 19

Finally create the HTML file, `DlgContent.html`, that is loaded in *FormCreate* above. This file will provide the content of our dialog box. *Listing 20* shows the HTML.

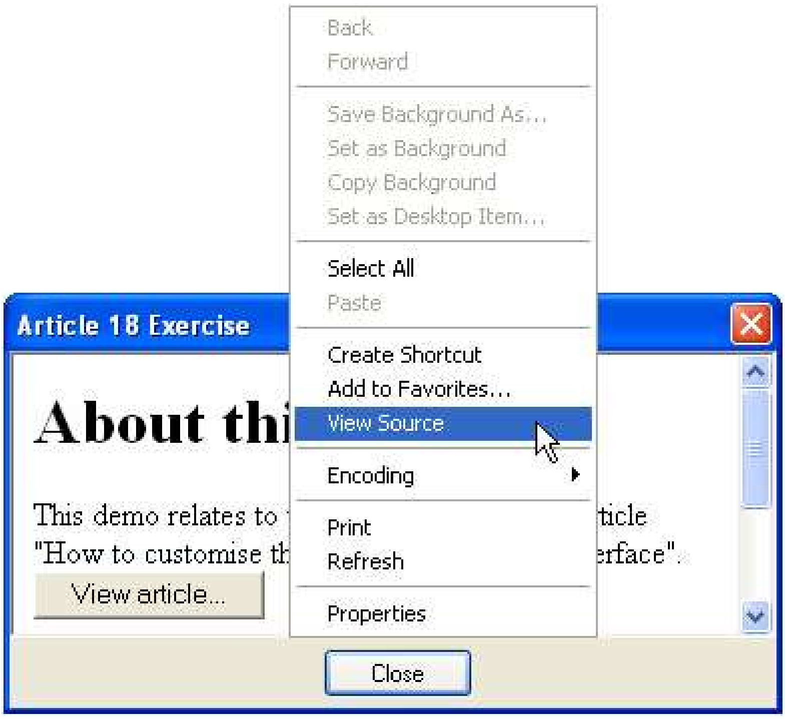
```
<html>
  <head>
    <title>Demo Dialog Content</title>
    <script type="text/javascript">
      function ViewArticle() {
        wdw = window.open();
        wdw.document.location =
          "http://www.delphidabbler.com/articles?article=18";
      }
    </script>
  </head>
  <body>
    <h1>
      About this demo
    </h1>
    <p>
      This demo relates to the DelphiDabbler.com article &quot;How to
      customise the TWebBrowser user interface&quot;.
      <input
        type="button"
        name="button"
        id="button"
        value="View article..."
        onclick="ViewArticle();"
      />
    </p>
    <p>
      &copy; Copyright P D Johnson
      (<a href="http://www.delphidabbler.com/"
      target="_blank">delphidabbler.com</a>), 2004-2006.
    </p>
    <p class="ruled">
      Right click above the line to see the custom pop-up menu.
    </p>
  </body>
</html>
```

Listing 20

This code defines a document which has:

- Some plain text including a paragraph styled using a CSS class named *.ruled* that is not defined in the document.
- A button that, when clicked, runs some JavaScript that opens a new window and displays this article in it.
- Some clickable text that opens a new browser window and navigates to the *DelphiDabbler.com* home page.

If you compile and run this application then right click in the browser control you should see something like this:



Doesn't look much like a dialog box does it? There are numerous problems:

1. The browser control is displaying its default border and scroll bar.
2. The HTML is being displayed in the default style.
3. The form's 'Close' button is themed while the 'View article' button displayed by the browser control is not.
4. Despite setting the *PopupMenu* property, the standard browser's context menu is still displayed.
5. Although it can't be seen here, you could select the text in the HTML document.

Let us now adapt the program to correct all these problems.

Stage 2

We will now utilize the classes we have developed in this article. To begin with add the units containing *IDocHOstUIHandler*, *TNulWBContainer* and *TWBContainer* to the project and refer to them in the main form's uses clause. Also add the `ActiveX` and `SysUtils` units to the uses clause.

It's now time to set up and use the container object. Switch to the form unit we developed in Stage 1 and add a field named *fWBContainer* of type *TWBContainer* to the form's class declaration. Now rewrite the form's *OnCreate* event handler as shown in *Listing 21*.

```
procedure TForm1.FormCreate(Sender: TObject);
const
  // Template for default CSS style
  cCSSTplt = 'body {'#13#10
    + '    background-color: %0:s;'#13#10
    + '    color: %1:s;'#13#10
    + '    font-family: "%2:s";'#13#10
    + '    font-size: %3:dpt;'#13#10
    + '    margin: 4px;'#13#10
    + '}'#13#10
    + 'h1 {'#13#10
    + '    font-size: %3:dpt;'#13#10
    + '    font-weight: bold;'#13#10
    + '    text-align: center;'#13#10
    + '}'#13#10
    + 'input#button {'#13#10
    + '    color: %1:s;'#13#10
    + '    font-family: "%2:s";'#13#10
    + '    font-size: %3:dpt;'#13#10
    + '}'#13#10
    + '.ruled {'#13#10
    + '    border-bottom: %4:s solid 2px;'#13#10
    + '    padding-bottom: 6px;'#13#10
    + '}';
var
  FmtCSS: string;  // Stores default CSS
begin
  // Create the CSS from system colours
  FmtCSS := Format(
    cCSSTplt,
    [ColorToHTML(Self.Color), ColorToHTML(Self.Font.Color),
    Self.Font.Name, Self.Font.Size,
    ColorToHTML(clInactiveCaption)]
  );
  // Create web browser container and set required properties
  fWBContainer := TWBContainer.Create(WebBrowser1);
  fWBContainer.UseCustomCtxMenu := True;    // use our popup menu
  fWBContainer.Show3DBorder := False;       // no border
  fWBContainer.ShowScrollBars := False;     // no scroll bars
  fWBContainer.AllowTextSelection := False; // no text selection
  fWBContainer.CSS := FmtCSS;               // CSS to be used
  // load content
  fWBContainer.HostedBrowser.Navigate(
    ExtractFilePath(ParamStr(0)) + 'DlgContent.html'
  );
end;
```

Listing 21

Key points to note in this method are:

- First we provide a template for a cascading style sheet we will use to make the HTML take on the appearance of the dialog box. This template includes some format strings as placeholders for colour and font information that will be filled in at run time.
- Next we fill in the required values in the CSS template and store the resulting code in *FmtCSS*. We get the colours and font from various properties of the form and from suitable system colours. Note that we use the *ColorToHTML* helper function (taken from the *Code Snippets Database*) to format colour information correctly:

```
function ColorToHTML(const Color: TColor): string;
var
  ColorRGB: Integer;
begin
  ColorRGB := ColorToRGB(Color);
  Result := Format(
    '#%0.2X%0.2X%0.2X',
    [GetRValue(ColorRGB), GetGValue(ColorRGB), GetBValue(ColorRGB)]
  );
end;
```

Listing 22

- Next we create the container object, passing the hosted *TWebBrowser* as a parameter, then set the properties as required.
- Finally we revert to the original code and load the document, except that we use the container object's *HostedBrowser* property to access the browser control rather than accessing it directly.

A couple more things remain to be done. The first is to handle the form's *OnDestroy* event to free the container object, as shown in *Listing 23*.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  fWBContainer.Free;  // free the container pbject
end;
```

Listing 23

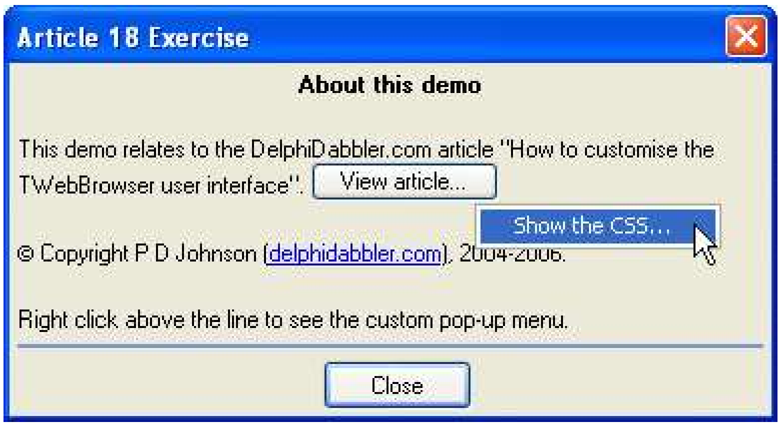
Finally we need to handle the *OnClick* event of our single context menu item. All we do here is display the default CSS in a message box, as *Listing 24* illustrates.

```
procedure TForm1.ShowtheCSS1Click(Sender: TObject);
begin
```

```
  ShowMessage(fWBContainer.CSS);  // display the CSS code
end;
```

Listing 24

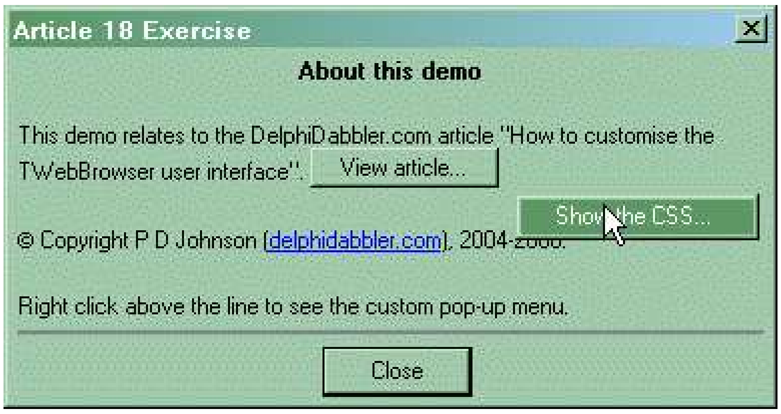
The moment of truth arrives. Let's run the revised application and see if it all works. Here's what we see on Windows XP with themes enabled:


Article 18 Exercise
About this demo
This demo relates to the DelphiDabbler.com article "How to customise the TWebBrowser user interface".
View article...
Show the CSS...
© Copyright P D Johnson (delphidabbler.com), 2004-2006.
Right click above the line to see the custom pop-up menu.
Close

Notice that:

- The borders and scroll bar have gone.
- The HTML is displayed in the correct dialog box font and colours.
- The 'View article' button is correctly themed.
- The paragraph with the *.ruled* CSS class now has a bottom border in the same colour as an inactive caption.
- Right clicking displays the single menu item of the *TPopupMenu* we assigned to the browser control's *PopupMenu* property.
- Take it on trust that you can't select text either!

Just to show that the browser control correctly adopts the current dialog box style, here is the same application running unchanged in on Windows XP using its classic style with the *Spruce* colour scheme:


Article 18 Exercise
About this demo
This demo relates to the DelphiDabbler.com article "How to customise the TWebBrowser user interface".
View article...
Show the CSS...
© Copyright P D Johnson (delphidabbler.com), 2004-2006.
Right click above the line to see the custom pop-up menu.
Close

All the code has now been completed. In the *final section* we will wrap up the article and provide a download containing all the source code for the sample application.