How to customise the TWebBrowser user interface (part 4 of 6)

Developing the customization class

Developing the customization class

As we decided above, *TWBContainer* will derive from *TNulWBContainer* and add all the required browser customization. Let us review what we want this customization to achieve:

1. Display either the browser's built in pop-up menu or the menu assigned to the *TWebBrowser.PopupMenu* property.
2. Display or hide 3D borders.
3. Display or hide scroll bars.
4. Customize document appearance at run time, via a custom cascading style sheet.
5. Allow or inhibit users from selecting text in the browser control.
6. The browser control will display themed controls (such as buttons) only when the host application is using themes.

We will define properties to configure the customization. We will then need to re-implement just two methods of *IDocHostUIHandler* to achieve the desired results – *ShowContextMenu* and *GetHostInfo*. *ShowContextMenu* controls the display of the popup menu and *GetHostInfo* enables the browser control's appearance to be customized.

Listing 12 shows the declaration of *TWBContainer*.

```
type
  TWBContainer = class(TBaseContainer,
    IDocHostUIHandler, IOleClientSite)
  private
    fUseCustomCtxMenu: Boolean;
    fShowScrollBars: Boolean;
    fShow3DBorder: Boolean;
    fAllowTextSelection: Boolean;
    fCSS: string;
  protected
    { Re-implemented IDocHostUIHandler methods }
    function ShowContextMenu(const dwID: DWORD;
      const ppt: PPOINT; const pcmdtReserved: IUnknown;
      const pdispReserved: IDispatch): HResult; stdcall;
    function GetHostInfo(var pInfo: TDocHostUIInfo): HResult; stdcall;
  public
    constructor Create(const HostedBrowser: TWebBrowser);
    property UseCustomCtxMenu: Boolean
      read fUseCustomCtxMenu write fUseCustomCtxMenu default False;
    property Show3DBorder: Boolean
      read fShow3DBorder write fShow3DBorder default True;
    property ShowScrollBars: Boolean
      read fShowScrollBars write fShowScrollBars default True;
    property AllowTextSelection: Boolean
      read fAllowTextSelection write fAllowTextSelection default True;
    property CSS: string
      read fCSS write fCSS;
  end;
```

Listing 12

Notice that we expose a property for each of the aspects of the browser control that we will customize, with the exception of theme support which we handle automatically. The properties are:

- *UseCustomCtxMenu* – the browser control displays its default context menu when the property is false and uses the menu assigned to *TWebBrowser PopupMenu* when true. If *UseContextMenu* is true but

PopupMenu is not assigned then no popup menu is displayed.

- *Show3DBorder* – the web browser displays a 3D border when the property is true and no border when false.
- *ShowScrollBars* – *TWebBrowser* displays scroll bars only if the property is true.
- *AllowTextSelection* – permits text selection in the browser control when true and inhibits it when false.
- *CSS* – provides the default cascading style sheet. This string property must contain valid CSS or be set to the empty string. When set we use this property to customize the document appearance.

The class constructor simply sets the default property values. These are chosen to leave the browser control in its default state. *Listing 13* shows the constructor.

```
constructor TWBContainer.Create(const HostedBrowser: TWebBrowser);
begin
  inherited;
  fUseCustomCtxMenu := False;
  fShowScrollBars := True;
  fShow3DBorder := True;
  fAllowTextSelection := True;
  fCSS := '';
end;
```

Listing 13

Now we come on to the meat of the code. First let's look at how we re-implement *ShowContextMenu*. This is easier to write than to describe. See *Listing 14* below.

```
function TWBContainer.ShowContextMenu(
  const dwID: DWORD;
  const ppt: PPOINT;
  const pcmdtReserved: IInterface;
  const pdispReserved: IDispatch): HResult;
begin
  if fUseCustomCtxMenu then
  begin
    // tell IE we're handling the context menu
    Result := S_OK;
    if Assigned(HostedBrowser.PopupMenu) then
      // browser has a pop up menu so activate it
      HostedBrowser.PopupMenu.Popup(ppt.X, ppt.Y);
  end
  else
    // tell IE to use default action: display own menu
    Result := S_FALSE;
end;
```

Listing 14

We first check the *fUseCustomCtxMenu* field to see what to do. If it is false we simply return *S_FALSE* to tell the web browser we have not handled the context menu. This causes the browser to display its default pop-up menu.

When *fUseCustomCtxMenu* is true we return *S_OK* to show we are handling the context menu ourselves. This prevents the default pop-up menu from being displayed. If the browser control's *PopupMenu* property is set we display the menu by calling its *Popup* method. The *ppt* parameter supplies the co-ordinates where the mouse was right clicked. We use this to position the top left corner of the pop-up menu.

GetHostInfo is more complex because it determines the display of the border, scroll bars, text selection, theme support and the default style sheet. We instruct the browser control about how to handle these items by filling in a *TDocHostUIInfo* structure, a pointer to which is passed as a parameter to the method. *Listing 15* defines this structure and comments describe its fields. *Listing 16* presents *GetHostInfo* itself.

```
type
  TDocHostUIInfo = record
    cbSize: ULONG;          // size of structure in bytes
    dwFlags: DWORD;         // flags that specify UI capabilities
    dwDoubleClick: DWORD;   // specified response to double click
    pchHostCss: PWChar;     // pointer to CSS rules
    pchHostNS: PWChar;      // pointer to namespace list for custom tags
  end;
```

Listing 15

```
function TWBContainer.GetHostInfo(
  var pInfo: TDocHostUIInfo): HResult;
const
  DOCHOSTUIFLAG_SCROLL_NO = $00000008;
  DOCHOSTUIFLAG_NO3DBORDER = $00000004;
  DOCHOSTUIFLAG_DIALOG = $00000001;
  DOCHOSTUIFLAG_THEME = $00040000;
  DOCHOSTUIFLAG_NOTHEME = $00080000;
begin
  try
    // Clear structure and set size
    ZeroMemory(@pInfo, SizeOf(TDocHostUIInfo));
    pInfo.cbSize := SizeOf(TDocHostUIInfo);
    // Set scroll bar visibility
    if not fShowScrollBars then
      pInfo.dwFlags := pInfo.dwFlags or DOCHOSTUIFLAG_SCROLL_NO;
    // Set border visibility
    if not fShow3DBorder then
      pInfo.dwFlags := pInfo.dwFlags or DOCHOSTUIFLAG_NO3DBORDER;
    // Decide if text can be selected
    if not fAllowTextSelection then
      pInfo.dwFlags := pInfo.dwFlags or DOCHOSTUIFLAG_DIALOG;
    // Ensure browser uses themes if application is doing
    if ThemeServices.ThemesEnabled then
      pInfo.dwFlags := pInfo.dwFlags or DOCHOSTUIFLAG_THEME
    else if ThemeServices.ThemesAvailable then
      pInfo.dwFlags := pInfo.dwFlags or DOCHOSTUIFLAG_NOTHEME;
    // Record default CSS as Unicode
    pInfo.pchHostCss := TaskAllocWideString(fCSS);
    if not Assigned(pInfo.pchHostCss) then
      raise Exception.Create(
        'Task allocator can''t allocate CSS string'
      );
    // Return S_OK to indicate we've made changes
    Result := S_OK;
  except
    // Return E_FAIL on error
    Result := E_FAIL;
  end;
end;
```

Listing 16

For our purposes we only need to use the *cbSize*, *dwFlags* and *pchHostCss* fields of *TDocHostUIInfo*. We set the remaining fields to zero. *cbSize* is set to the size of the structure – a common Windows idiom.

dwDoubleClick

This *TDocHostUIInfo* field is ignored by some browser versions.

Next we decide which flags to store in *dwFlags* to customize the control's user interface capabilities. Which flags we specify depends on the state of the *ShowScrollBars*, *Show3DBorder* and *AllowTextSelection* properties and whether themes are enabled. The flags we use are:

- *DOCHOSTUIFLAG_SCROLL_NO* prevents the vertical scroll bar from being displayed.
- *DOCHOSTUIFLAG_NO3DBORDER* inhibits the display of 3D borders.
- Slightly less obviously, *DOCHOSTUIFLAG_DIALOG* prevents text selection – the name comes from the main use of this flag to prevent text selection in dialog boxes.

Other flags

There are numerous other *DOCHOSTUIFLAG_* flags that can be assigned to *dwFlags*. The full set is defined in this article's *demo code*.

- More complex is the way we ensure that the application's themes are echoed by the web browser control. We use the `Themes` unit's *ThemeServices* object to check if themes are enabled. If so we use the *DOCHOSTUIFLAG_THEME* flag to request that the browser also uses themes. If themes are not enabled, but available, we switch off the browser's theme support by using

DOCHOSTUIFLAG_NOTHEME. If themes are not available at all (e.g. in Windows 2000) we do nothing.

Lastly, we store the default style sheet (per the *CSS* property) as Unicode text in the *pchHostCss* field. We have to allocate storage for this Unicode string – and we **must** use the task allocator to do this. Note that we are not responsible for freeing this memory – *TWebBrowser* does this.

We use the *TaskAllocWideString* helper function (taken from the *Code Snippets Database*) to allocate the storage and ensure the string is in Unicode format. The routine is shown in *Listing 17*. Note that this code works on both Unicode and non-Unicode Delphis.

```
function TaskAllocWideString(const S: string): PWChar;
var
  StrLen: Integer;  // length of string in bytes
begin
  // Store length of string in characters, allowing for terminal #0
  StrLen := Length(S) + 1;
  // Allocate buffer for wide string using task allocator
  Result := CoTaskMemAlloc(StrLen * SizeOf(WideChar));
  if Assigned(Result) then
    // Convert string to wide string and store in buffer
    StringToWideChar(S, Result, StrLen);
end;
```

Listing 17

And that completes the browser customization code. If you need to specify different customizations simply define a new class that descends from *TNulWBContainer* and provide the required functionality by re-implementing the appropriate *IDocHostUIHandler* methods.

Other examples

Article #22: "How to call Delphi code from scripts running in a TWebBrowser" demonstrates another class that descends from *TNulWBContainer*, this time re-implementing the *GetExternal* method.

To use our customization class simply create an instance of *TWBContainer*, passing a reference to the browser control that is to be customized, set the required properties of *TWBContainer* then load the required document.

Note

You should always set the properties of *TWBContainer* before loading any document into the browser control. The reason for this is that on operating systems earlier than Windows XP SP2 the control only reads the default CSS when the first document is loaded. Changing the *CSS* property after loading the first document will have no effect.

In the *next section* we will test our code by creating a sample application that exercises *TWBContainer*.