COM Corner: ActiveX Documents, Part 2

by Steve Teixeira

Last month we had some fun building a very basic ActiveX Document Server in Delphi. As I promised then, we're now going to delve deeper into the world of ActiveX Documents. In particular, we'll extend last month's `TDelphi AxDoc` so that it will support being hosted in MS Word or Internet Explorer and handles activation issues and menu merges.

Just to make things a little more interesting, I've chosen to make the ActiveX Document Server into an out-of-process server (.EXE) rather than an in-process server (.DLL). I accomplished this in part by of course changing `library` to `program` in the .DPR file. When invoked standalone, the application shows the main form containing a `TMemo` and a menu. When invoked via Automation, the main form does not show and the ActiveX Document can be embedded in the container application invoking the server. I played a little trick in the beginning of the .DPR file to prevent the application's main form showing when the application is invoked via Automation. I did this by setting `Application.ShowMainForm` to the appropriate value, depending on the value set for `ComServer.StartMode` (Listing 1).

➤ *Listing 1*

```
begin
  Application.Initialize;
  // Don't show main form when
  // started via automation
  Application.ShowMainForm :=
    ComServer.StartMode =
      smStandAlone;
  Application.CreateForm(
    TFormMain, FormMain);
  Application.Run;
end.
```

Registration

To support embedding in Word and Internet Explorer (IE) an ActiveX Document Server must make a number of registry entries that aren't required if you're working strictly from the spec. This shouldn't surprise veteran COM developers, who have probably learned by now that you'll get very little to function correctly in COM if you do it strictly by the spec! Try to keep in your mind that existing application implementation is often of more importance than the spec. In this case, I learned that an `InprocHandler32` key must be made under the object's `CLSID` and `DocObject` and `Insertable` keys must be made under the object's `ProgID`. The source code for my factory's `UpdateRegistry` method, which handles all the registration duties, is in Listing 2.

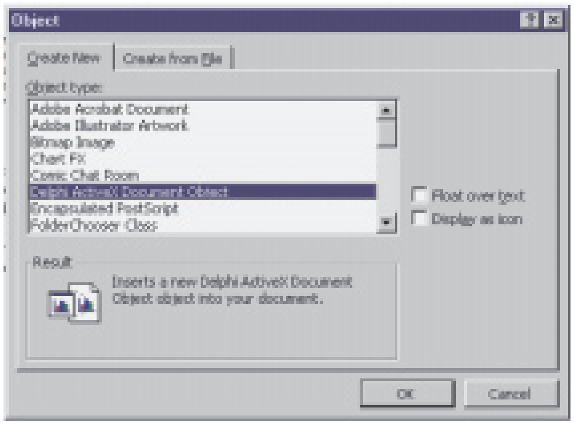
You might be unfamiliar with the `InprocHandler32` entry, because it's used primarily in the context of ActiveX Documents. `InProcHandler32` specifies the in-process handler for this server. An in-process handler is structurally identical to an in-process server, except it is designed to serve a different purpose. Think of in-process handlers as lightweight entities that may provide some services to a client, but will delegate to the out-of-process server specified in `LocalServer32` for any 'real' work. Developers can create custom handlers for their servers or specify the default in-process handler, which is Ole32.dll.

Once you get your server registered correctly, you will be able to see it in Word's `Insert Object` dialog found by selecting `Insert | Object` from the main menu. This dialog is shown in Figure 1.

Framework

While getting your ActiveX Document server to show up in the nice little dialog can certainly be

➤ *Figure 1*



Inserts a new Delphi ActiveX Document Object object into your document.

➤ *Listing 2*

```
procedure TActiveXDocumentFactory.UpdateRegistry(
  Register: Boolean);
var ClassKey, ProgKey, MiscFlags: string;
begin
  ClassKey := 'CLSID\' + GUIDToString(ClassID) + '\';
  ProgKey := ProgID + '\';
  if Register then begin
    inherited UpdateRegistry(Register);
    MiscFlags := IntToStr(FDocMiscStatus);
    // Add reg keys under CLSID
    CreateRegKey(ClassKey+'DocObject', '', MiscFlags);
    CreateRegKey(ClassKey+'Programmable', '', '');
    CreateRegKey(ClassKey+'Insertable', '', '');
    CreateRegKey(ClassKey+'InprocHandler32', '', FHandler);
    // Add reg keys under ProgID
    CreateRegKey(ProgKey+'DocObject', '', MiscFlags);
    CreateRegKey(ProgKey+'Insertable', '', '');
    // Remove "control" key added by inherited method
    DeleteRegKey(ClassKey + 'Control');
  end else begin
    DeleteRegKey(ClassKey + 'DefaultExtension');
    DeleteRegKey(ClassKey + 'DefaultIcon');
    DeleteRegKey(ClassKey + 'DocObject');
    DeleteRegKey(ClassKey + 'Programmable');
    DeleteRegKey(ClassKey + 'Insertable');
    DeleteRegKey(ClassKey + 'InprocHandler32');
    DeleteRegKey(ProgKey + 'DocObject');
    DeleteRegKey(ProgKey + 'Insertable');
    inherited UpdateRegistry(Register);
  end;
end;
```

```
function TActiveXDocument.InPlaceDeactivate: HResult;
var
  ParentWnd: HWND;
begin
  // This is a work-around for the fact that TActiveXControl implementation of
  // this method makes the control go away to ParkingWindow la-la land.  It
  // needs to stay put within the document.
  ParentWnd := Control.ParentWindow;
  Result := inherited InplaceDeactivate;
  Control.ParentWindow := ParentWnd;
  Control.Visible := True;
end;
```

➤ *Listing 3*

considered a minor victory, it has no bearing on whether or not the server will function properly when called upon, so there's more work ahead. Before I discuss some of the framework I built to support ActiveX Documents on top of VCL's Delphi ActiveX control (DAX) framework, I feel compelled to provide a disclaimer. As one of the designers of DAX, I can tell you with certainty that DAX was never intended to support ActiveX Documents. This fact will be highlighted when you witness some of the stunts it was necessary to pull in the base `TActiveXDocument` class in order to get things clicking. That isn't inherently a Bad Thing because, given the alternative of reimplementing a gazillion interfaces that are already implemented 99% correctly in `TActiveXControl`, I think you'll agree that a few hacks here and there to retrofit support on a nearly compatible foundation is better than starting from scratch. Also, as a developer who measures productivity by the lines of code that I *don't* write, it's kind of satisfying to enhance an existing framework and not reinvent a lot of wheels. I'll point out these stunts as we go along.

Last month we saw that support for `IOleDocument` and `IOleDocumentView` (and many other interfaces) is necessary for a proper ActiveX Document Server, and these were implemented in the `TActiveXDocument` class. However, you may have noticed that activation and deactivation behavior was a little flaky, because most of the work associated with this is being done within DAX. For example, the standard behavior for an ActiveX control when it becomes deactivated is to hide itself and re-parent itself to an internal 'parking' window. However, this behavior doesn't work so well for ActiveX documents, which should be visible in the container even when they are not active. We can work around this problem by reimplementing the `InPlaceDeactivate` method of `IOleInPlaceObject`. It's quite simple to reimplement a single method of an interface that is implemented by an ancestor class: all you need to do is add the interface to the inheritance list and implement the new method. For example, the inheritance list for `TActiveXDocument` would now look like this:

```
TActiveXDocument =  class(
  TActiveXControl, IOleDocument,
  IOleDocumentView,
  IOleInPlaceObject)
```

And the method would be reimplemented in the declaration for this object as:

```
function InPlaceDeactivate:
  HResult; stdcall;
```

Since we are only implementing one method on this interface, the compiler will bind to the ancestor's implementation of the other methods to complete the interface implementation. This `InPlaceDeactivate` method is implemented as shown in Listing 3.

You can see that this code resets the parent window to the original value and ensures the control is visible after calling the inherited implementation. Not the optimal algorithm, but the server relies on behavior living within the `TActiveXControl` implementation of this method, so this is an acceptable real-world compromise.

The merging of the server's menus with those of the container needs to occur whenever the server is activated. Therefore, there are a couple of paths to activation that the server must hook into in order to provide proper menu merging behavior. The first is `IOleInPlaceActiveObject.OnDocWindowActivate`. This interface is implemented in `TActiveXControl`, so, like the previous issue, we can reimplement only the needed method to add our own logic as shown in Listing 4.

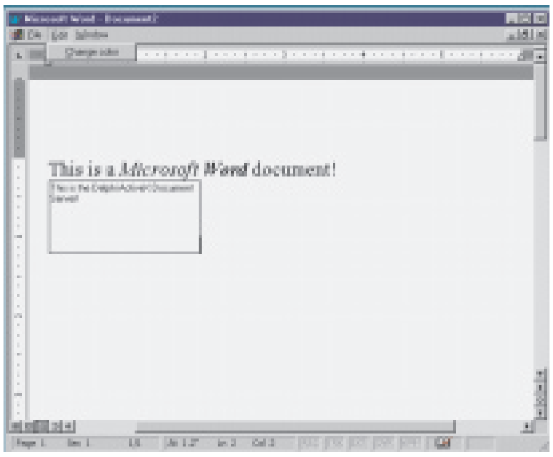
There is another path to activation that is much trickier to intercept. Several locations within the DAX framework call a helper function called `InPlaceActivate` to perform activation housekeeping. That makes this method an ideal location to consolidate the call to merge menus. Unfortunately, this method is defined as static, so it can't be reimplemented like an interface method or overridden like a virtual. Therefore, the only recourse in this case is to add a copy of the AxCtrls.pas unit to my project and modify my copy of that unit, making the `TActiveXControl.InPlaceActivate` method `virtual`. Generally speaking, you want to avoid having to modify the VCL source code if at all possible, but there are cases such as this where a more 'VCL-friendly' approach doesn't exist. After marking this method as `virtual`, I am able to

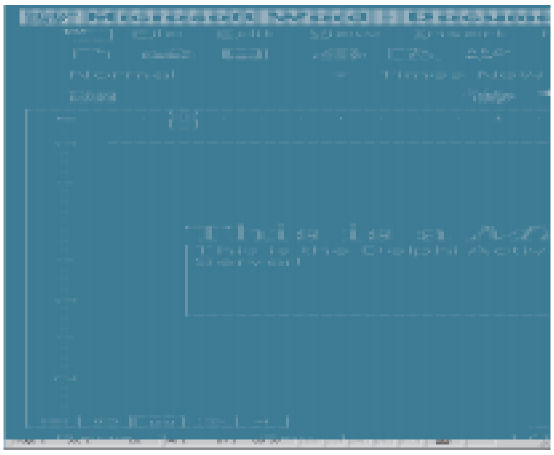
```
function TActiveXDocument.OnDocWindowActivate(fActivate: BOOL): HResult;
begin
  Result := inherited OnDocWindowActivate(fActivate);
  if fActivate then
    InPlaceMenuCreate
  else
    InPlaceMenuDestroy;
end;
```

➤ *Above: Listing 4*

➤ *Below: Listing 5*

```
function TActiveXDocument.InPlaceActivate(ActivateUI: Boolean): HResult;
begin
  Result := inherited InPlaceActivate(ActivateUI);
  InPlaceMenuCreate;
end;
```


This is a Microsoft Word document!
This is the Delphi ActiveX Document Server!



➤ Top: Figure 2

➤ Bottom: Figure 3

➤ Facing page: Listing 6

override it in the TActiveXDocument class as shown in Listing 5.

You may have noticed that the previous two methods employ procedures called InPlaceMenuCreate and InPlaceMenuDestroy. These contain the logic for creating and managing the merged menus as displayed in the container application, and are shown in Listing 6.

Listing 7 shows the Main.pas unit, which contains this specific implementation of a TActiveX Document-based ActiveX Document Server. This unit overrides the streaming mechanism of the server so that it streams out only a simple text stream of the memo contents. This unit also takes responsibility for creating the menu that will eventually be merged into container frames.

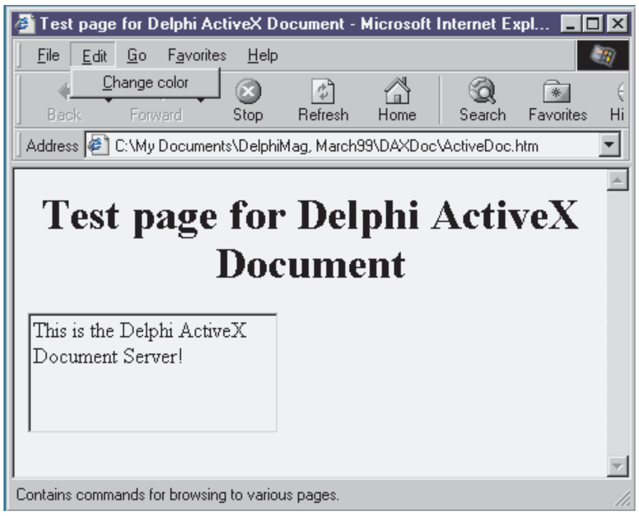
Testing The Server

After building the server, remember to register it by running it once or using the /regserver command line switch. Once registered, it's time to test it in our target containers. The Delphi ActiveX Document server can be inserted into a Word document as I described earlier, using the Insert Object dialog found on the Insert | Object menu. Figure 2 shows the Delphi ActiveX Document Server embedded in an MS Word document. Notice that the menus have merged and the toolbars have been hidden to accommodate the server while it is active. When I click outside the server, Word's menus and toolbars return as shown in Figure 3.

The process for embedding the ActiveX Document in a web page is rather different: you must include an OBJECT tag in your HTML file that references the CLSID of the server. For example, the HTML code below could serve as a test bed for the Delphi ActiveX Document server:

```
<HTML><HEAD><TITLE>
Test page for Delphi ActiveX
Document</TITLE></HEAD>
<BODY><center><h1>Test page
for Delphi ActiveX Document
</h1></center>
<OBJECT ID="DAXDoc.DelphiAxDoc"
<CLASSID="CLSID:C10BE34F-A81F-
11D2-AF31-0000861EF0BB">>
</OBJECT></BODY></HTML>
```

Viewing this HTML in Internet Explorer 4, you will see a web page containing a message at the top and the server embedded in the document immediately below the message, as shown in Figure 4. Notice that IE 4 has also merged the doc server's menu with its own main menu.


Test page for Delphi ActiveX Document - Microsoft Internet Expl...
File
Edit
Go
Favorites
Help
Change color
Back
Forward
Stop
Refresh
Home
Search
Favorites
Hi
Address
C:\My Documents\DelphiMag, March99\DAXDoc\ActiveDoc.htm

Test page for Delphi ActiveX Document

This is the Delphi ActiveX Document Server!

Contains commands for browsing to various pages.

➤ Figure 4

Summary

Over the past two instalments of *COM Corner*, you have learned the ins and outs of ActiveX Document servers and how to use the VCL's DAX framework to jump start your way to writing ActiveX Document Servers.

ActiveX Document Servers provide a powerful way to enable users to edit your application's documents within the bounds of ActiveX Document containers such as Visual Basic, Word and Internet Explorer. Not only that, but ActiveX Document Servers also provide a means for you to take advantage of OLE's embedding and structured storage technologies to include your own type of documents in some other context. ActiveX Document Servers build on many different COM technologies, so an understanding of how they work means an understanding of much of the fundamentals of COM.

See you next time for more nuggets from the fascinating world of COM development...!

Steve Teixeira is Vice President of Software Development for US-based DeVries Data Systems, a provider of consulting services for Inprise and Microsoft tools and technologies. If you have a great idea that you would like to see become a *COM Corner* article, email Steve at steve@dvdata.com

```
{ActiveX Document Support Unit. Copyright © 1999 Steve Teixeira}
unit AxDocs;
interface
uses
  Windows, ComObj, ActiveX, AxCtrls, Controls, Classes,
  Menus, Messages;
type
  TActiveXDocumentFactory = class;
  TActiveXDocument = class(TActiveXControl, IOleDocument,
    IOleDocumentView, IOleInPlaceActiveObject,
    IOleInPlaceObject)
  private
    FFactory: TActiveXDocumentFactory;
    FMenu: TMainMenu;
    FOleMenu: HMENU;
    FSharedMenu: HMENU;
    function GetAncestorValueByField(FieldNum: Cardinal):
      Cardinal;
    procedure SetAncestorValueByField(FieldNum,
      Value: Cardinal);
    function GetOleInPlaceSite: IOleInPlaceSite;
    procedure SetOleInPlaceSite(
      const Value: IOleInPlaceSite);
    procedure InPlaceMenuCreate;
    procedure InPlaceMenuDestroy;
    procedure MergeMenus(SharedMenu, SourceMenu: HMENU;
      MenuWidths: PInteger; WidthIndex: Integer);
    procedure UnmergeMenus(SharedMenu, SourceMenu: HMENU);
      protected
    { IOleDocument methods }
    function CreateView(Site: IOleInPlaceSite; Stream:
      IStream; rsrvd: DWORD; out View: IOleDocumentView):
      HResult; stdcall;
    function GetDocMiscStatus(var Status: DWORD): HResult;
      stdcall;
    function EnumViews(out Enum: IEnumOleDocumentViews;
      out View: IOleDocumentView):HResult; stdcall;
    { IOleDocumentView methods }
    function SetInPlaceSite(Site: IOleInPlaceSite): HResult;
      stdcall;
    function GetInPlaceSite(out Site: IOleInPlaceSite):
      HResult; stdcall;
    function GetDocument(out P: IUnknown): HResult; stdcall;
    function SetRect(const View: TRECT): HResult; stdcall;
    function GetRect(var View: TRECT): HResult; stdcall;
    function SetRectComplex(const View, HScroll, VScroll,
      SizeBox): HResult; stdcall;
    function Show(fShow: BOOL): HResult; stdcall;
    function UIActivate(fUIActivate: BOOL): HResult;
      stdcall;
    function Open: HResult; stdcall;
    function CloseView(dwReserved: DWORD): HResult; stdcall;
    function SaveViewState(pstm: IStream): HResult; stdcall;
    function ApplyViewState(pstm: IStream): HResult;
      stdcall;
    function Clone(NewSite: IOleInPlaceSite; out NewView:
      IOleDocumentView):HResult; stdcall;
    { IOleInPlaceActiveObject }
    function OnDocWindowActivate(fActivate: BOOL): HResult;
      stdcall;
    { IOleInPlaceObject }
    function InPlaceDeactivate: HResult; stdcall;
    { Overrides }
    procedure GetDocUIInfo(var Menu: TMainMenu);
    function InPlaceActivate(ActivateUI: Boolean): HResult;
      override;
    procedure WndProc(var Message: TMessage); override;
  public
    procedure Initialize; override;
    function ObjQueryInterface(const IID: TGUID; out Obj):
      HResult; override;
    property Menu: TMainMenu read FMenu write FMenu;
    property OleInPlaceSite: IOleInPlaceSite
      read GetOleInPlaceSite write SetOleInPlaceSite;
  end;
  TActiveXDocClass = class of TActiveXDocument;
  TActiveXDocumentFactory = class(TActiveXControlFactory)
  private
    FDocMiscStatus: DWORD;
    FHandler: string;
  public
    property DocMiscStatus: DWORD read FDocMiscStatus;
    constructor Create(ComServer: TComServerObject;
      ActiveXDocClass: TActiveXDocClass;
      WinControlClass: TWinControlClass;
      const ClassID: TGUID;
      ToolboxBitmapID, MiscStatus: Integer;
      ThreadingModel: TThreadingModel;
      const Handler: string; DocMiscStatus: DWORD);
    procedure UpdateRegistry(Register: Boolean); override;
  end;
implementation
uses
  ComServ, SysUtils, Forms;

{ ** LISTING CONTINUES ON FOLLOWING PAGE ** }
```

```
function TActiveXDocument.ObjQueryInterface(
  const IID: TGUID; out Obj): HResult;
begin
  // Must stub out IOleLink, or container will assume this
  // is a linked object rather than an embedded object.
  if IsEqualGuid(IID, IOleLink) then
    Result := E_NOINTERFACE
  else
    Result := inherited ObjQueryInterface(IID, Obj);
end;
function TActiveXDocument.GetOleInPlaceSite:
  IOleInPlaceSite;
begin
  // Work around fact that FOleInPlaceSite is private in
  // TActiveXControl. Only guaranteed to work in Delphi 4
  Result := IOleInPlaceSite(GetAncestorValueByField(9));
end;
procedure TActiveXDocument.SetOleInPlaceSite(
  const Value: IOleInPlaceSite);
begin
  // Workaround, as above
  SetAncestorValueByField(9, Cardinal(Value));
end;
function TActiveXDocument.GetAncestorValueByField(
  FieldNum: Cardinal): Cardinal;
var ParentInstanceSize, Ofs: Cardinal;
begin
  // Nasty hack: this method returns the value of a
  // particular field in ancestor class, with assumption
  // that given field and all prior fields are size 4 bytes
  ParentInstanceSize :=
    ClassParent.ClassParent.InstanceSize;
  Ofs := ParentInstanceSize + ((FieldNum - 1) * 4);
  asm
    mov eax, Self
    add eax, Ofs
    mov eax, dword ptr [eax]
    mov @Result, eax
  end;
end;
procedure TActiveXDocument.SetAncestorValueByField(
  FieldNum, Value: Cardinal);
var ParentInstanceSize, Ofs: Cardinal;
begin
  // Nasty hack: as above
  ParentInstanceSize :=
    ClassParent.ClassParent.InstanceSize;
  Ofs := ParentInstanceSize + ((FieldNum - 1) * 4);
  asm
    mov eax, Self
    add eax, Ofs
    mov ecx, Value
    mov dword ptr [eax], ecx
  end;
end;
procedure TActiveXDocument.Initialize;
begin
  inherited Initialize;
  FFactory := Factory as TActiveXDocumentFactory;
end;
procedure TActiveXDocument.GetDocUIInfo(var Menu:
TMainMenu);
begin
  Menu := nil;
end;
function TActiveXDocument.InPlaceActivate(
  ActivateUI: Boolean): HResult;
begin
  Result := inherited InPlaceActivate(ActivateUI);
  InPlaceMenuCreate;
end;
procedure TActiveXDocument.WndProc(var Message: TMessage);
begin
  inherited WndProc(Message);
  if Message.Msg = WM_LBUTTONDBLCLK then
    InPlaceActivate(True);
end;
procedure TActiveXDocument.InPlaceMenuCreate;
var
  IPFrame: IOleInPlaceFrame;
  IPSite: IOleInPlaceSite;
  IPUIWindow: IOleInPlaceUIWindow;
  omgw: TOleMenuGroupWidths;
  FrameInfo: TOleInPlaceFrameInfo;
  PosRect, ClipRect: TRect;
begin
  OleCheck(ClientSite.QueryInterface(
    IOleInPlaceSite, IPSite));
  FrameInfo.cb := sizeof(FrameInfo);
  IPSite.GetWindowContext(IPFrame, IPUIWindow, PosRect,
    ClipRect, FrameInfo);
  FillChar(omgw, SizeOf(omgw), 0);
  // Create a blank menu and ask the container to add its
  // menus into the TOleMenuGroupWidths record
  FSharedMenu := CreateMenu;
  try
    OleCheck(IPFrame.InsertMenus(FSharedMenu, omgw));
    if FMenu = nil then Exit;
    MergeMenus(FSharedMenu, FMenu.Handle, @omgw.width, 1);
    // Send the menu to the client
    FOleMenu := OleCreateMenuDescriptor(FSharedMenu, omgw);
    IPFrame.SetMenu(FSharedMenu, FOleMenu, Control.Handle);
  except
    DestroyMenu(FSharedMenu);
    FSharedMenu := 0;
    raise;
  end;
end;
procedure TActiveXDocument.InPlaceMenuDestroy;
var
  IPFrame: IOleInPlaceFrame;
  IPSite: IOleInPlaceSite;
  IPUIWindow: IOleInPlaceUIWindow;
  FrameInfo: TOleInPlaceFrameInfo;
  PosRect, ClipRect: TRect;
begin
  // Get the clients IOleInPlaceFrame so we can ask
  // it to remove it's menu
  OleCheck(ClientSite.QueryInterface(
    IOleInPlaceSite, IPSite));
  FrameInfo.cb := sizeof(FrameInfo);
  IPSite.GetWindowContext(IPFrame, IPUIWindow, PosRect,
    ClipRect, FrameInfo);
  if IPFrame <> nil then
    IPFrame.SetMenu(0, 0, 0);
  OleDestroyMenuDescriptor(FOleMenu);
  FOleMenu := 0;
  UnmergeMenus(FSharedMenu, FMenu.Handle);
end;
type
  PIntArray = ^TIntArray;
  TIntArray = array[0..0] of Integer;
procedure TActiveXDocument.MergeMenus(
  SharedMenu, SourceMenu: HMENU; MenuWidths: PInteger;
  WidthIndex: Integer);
var
  MenuItems, GroupWidth, Position, I, Len: Integer;
  MenuState: UINT;
  PopupMenu: HMENU;
  ItemText: array[0..255] of char;
begin
  // Copy the popups from the pMenuSource
  MenuItems := GetMenuItemCount(SourceMenu);
  GroupWidth := 0;
  Position := 0;
  // Insert at appropriate spot depending on WidthIndex
  if (WidthIndex < 0) or (WidthIndex > 1) then
    Exit;
  if WidthIndex = 1 then
    Position := MenuWidths^;
  for I := 0 to MenuItems - 1 do begin
    // Get the HMENU of the popup
    PopupMenu := GetSubMenu(SourceMenu, I);
    // Separators move us to next group
    MenuState := GetMenuState(SourceMenu, I, MF_BYPOSITION);
    if (PopupMenu = NULL) and
      ((MenuState and MF_SEPARATOR) <> 0) then begin
      if WidthIndex > 5 then
        Exit;   // Servers should not touch past 5
      PIntArray(MenuWidths)^[WidthIndex] := GroupWidth;
      GroupWidth := 0;
      if WidthIndex < 5 then
        Inc(Position, PIntArray(MenuWidths)^[WidthIndex+1]);
      Inc(WidthIndex, 2);
    end else begin
      // Get the menu item text
      Len := GetMenuString(SourceMenu, I, ItemText,
        SizeOf(ItemText), MF_BYPOSITION);
      // Popups handled differently to normal menu items
      if PopupMenu <> 0 then begin
        if GetMenuItemCount(PopupMenu) <> 0 then begin
          // Strip HIBYTE because contains a count of items
          MenuState := LoByte(MenuState) or MF_POPUP;
          // Non-empty popup, add it to shared menu bar
          InsertMenu(SharedMenu, Position, MenuState or
            MF_BYPOSITION, PopupMenu, ItemText);
          Inc(Position);
          Inc(GroupWidth);
        end;
      end
      else if Len > 0 then begin
        // only non-empty items are added
        if ItemText <> '' then begin
          // here state doesn’t contain a count in  HIBYTE
          InsertMenu(SharedMenu, Position,
            MenuState or MF_BYPOSITION,
            GetMenuItemID(SourceMenu, I), ItemText);
          Inc(Position);
          Inc(GroupWidth);
        end;
      end;
    end;
  end;
end;
procedure TActiveXDocument.UnmergeMenus(
  SharedMenu, SourceMenu: HMENU);
var
  TheseItems, MenuItems, I, J: Integer;
  PopupMenu: HMENU;
begin
  MenuItems := GetMenuItemCount(SharedMenu);
  TheseItems := GetMenuItemCount(SourceMenu);
```

```
    for I := MenuItems - 1 downto 0 do begin
      // Check the popup menus
      PopupMenu := GetSubMenu(SharedMenu, I);
      if PopupMenu <> 0 then begin
        // If it is one of ours, remove it from the SharedMenu
        for J := 0 to TheseItems - 1 do begin
          if GetSubMenu(SourceMenu, J) = PopupMenu then begin
            // Remove the menu from SharedMenu
            RemoveMenu(SharedMenu, I, MF_BYPOSITION);
            Break;
          end;
        end;
      end;
    end;
  end;
{ TActiveXDocument.IOleDocument }
function TActiveXDocument.CreateView(Site: IOleInPlaceSite;
  Stream: IStream; rsrvd: DWORD;
  out View: IOleDocumentView): HResult;
var OleDocView: IOleDocumentView;
begin
  Result := S_OK;
  try
    if View = nil then begin
      Result := E_POINTER;
      Exit;
    end;
    OleDocView := Self as IOleDocumentView;
    if (OleInPlaceSite = nil) or
      (OleDocView = nil) then begin
      Result := E_FAIL;
      Exit;
    end;
    // Use site provided
    if Site <> nil then
      OleDocView.SetInPlaceSite(Site);
    // Use stream provided for initialization
    if Stream <> nil then
      OleDocView.ApplyViewState(Stream);
    View := OleDocView;    // Return the view
  except
    Result := E_FAIL;
  end;
end;
function TActiveXDocument.EnumViews(
  out Enum: IEnumOleDocumentViews;
  out View: IOleDocumentView): HResult;
begin
  Result := S_OK;
  try
    View := Self as IOleDocumentView;
  except
    Result := E_FAIL;
  end;
end;
function TActiveXDocument.GetDocMiscStatus(
  var Status: DWORD): HResult;
begin
  Status :=
    (Factory as TActiveXDocumentFactory).DocMiscStatus;
  Result := S_OK;
end;
{ TActiveXDocument.IOleDocument }
function TActiveXDocument.ApplyViewState(
  pstm: IStream): HResult;
begin
  Result := E_NOTIMPL;
end;
function TActiveXDocument.Clone(NewSite: IOleInPlaceSite;
  out NewView: IOleDocumentView): HResult;
begin
  Result := E_NOTIMPL;
end;
function TActiveXDocument.CloseView(
  dwReserved: DWORD): HResult;
begin
  Result := S_OK;
  try
    Show(False);
    SetInPlaceSite(nil);
  except
    Result := E_UNEXPECTED;
  end;
end;
function TActiveXDocument.GetDocument(
  out P: IUnknown): HResult;
begin
  Result := S_OK;
  try
    P := Self as IUnknown;
  except
    Result := E_FAIL;
  end;
end;
function TActiveXDocument.GetInPlaceSite(
  out Site: IOleInPlaceSite): HResult;
begin
  Result := S_OK;
  try
    Site := OleInPlaceSite;
  except
    Result := E_FAIL;
  end;
end;
function TActiveXDocument.GetRect(var View: TRECT): HResult;
begin
  Result := S_OK;
  try
    View := Control.BoundsRect;
  except
    Result := E_UNEXPECTED;
  end;
end;
function TActiveXDocument.Open: HResult;
begin
  Result := E_NOTIMPL;
end;
function TActiveXDocument.SaveViewState(pstm: IStream):
HResult;
begin
  Result := E_NOTIMPL;
end;
function TActiveXDocument.SetInPlaceSite(
  Site: IOleInPlaceSite): HResult;
begin
  Result := S_OK;
  try
    if OleInPlaceSite <> nil then
      Result := InPlaceDeactivate;
    if Result <> S_OK then
      Exit;
    if Site <> nil then
      OleInPlaceSite := Site;
  except
    Result := E_UNEXPECTED;
  end;
end;
function TActiveXDocument.SetRect(
  const View: TRECT): HResult;
begin
  // Implement using TActiveXControl's
  // IOleInPlaceObject.SetObjectRects
  Result := SetObjectRects(View, View);
end;
function TActiveXDocument.SetRectComplex(const View; const
  HScroll; const VScroll; const SizeBox): HResult;
begin
  Result := E_NOTIMPL;
end;
function TActiveXDocument.Show(fShow: BOOL): HResult;
begin
  try
    if fShow then
      Result := InPlaceActivate(False)
    else begin
      Result := UIActivate(False);
      Control.Visible := False;
    end;
  except
    Result := E_UNEXPECTED;
  end;
end;
function TActiveXDocument.UIActivate(
  fUIActivate: BOOL): HResult;
begin
  Result := S_OK;
  try
    if FUIActivate then begin
      if OleInPlaceSite <> nil then
        InPlaceActivate(True)
      else
        Result := E_UNEXPECTED;
    end else begin
      UIDeactivate;
      InPlaceMenuDestroy;
    end;
  except
    Result := E_UNEXPECTED;
  end;
end;
{ TActiveXDocument.IOleInPlaceActiveObject }
function TActiveXDocument.OnDocWindowActivate(
  fActivate: BOOL): HResult;
begin
  Result := inherited OnDocWindowActivate(fActivate);
  if fActivate then InPlaceMenuCreate
  else InPlaceMenuDestroy;
end;
{ TActiveXDocument.IOleInPlaceObject }
function TActiveXDocument.InPlaceDeactivate: HResult;
var ParentWnd: HWND;
begin
  // This is a work-around for the fact that TActiveXControl
  // implementation of this method makes the control go away
  // to ParkingWindow la-la land. It needs to stay put
  // within the document.
  ParentWnd := Control.ParentWindow;
  Result := inherited InplaceDeactivate;
  Control.ParentWindow := ParentWnd;
  Control.Visible := True;
end;
{ TActiveXDocumentFactory }
constructor TActiveXDocumentFactory.Create(
  ComServer: TComServerObject;
  ActiveXDocClass: TActiveXDocClass;
  WinControlClass: TWinControlClass;
  const ClassID: TGUID;
  ToolboxBitmapID, MiscStatus: Integer;
  ThreadingModel: TThreadingModel;
  const Handler: string; DocMiscStatus: DWORD);
begin
  FDocMiscStatus := DocMiscStatus;
  if Handler <> '' then
    FHandler := Handler
  else
    FHandler := 'ole32.dll';
  inherited Create(ComServer, ActiveXDocClass,
    WinControlClass, ClassId, ToolboxBitmapID, '',
    MiscStatus, ThreadingModel);
end;
procedure TActiveXDocumentFactory.UpdateRegistry(
  Register: Boolean);
var
  ClassKey, ProgKey, MiscFlags: string;
begin
  ClassKey := 'CLSID\' + GUIDToString(ClassID) + '\';
  ProgKey := ProgID + '\';
  if Register then begin
    inherited UpdateRegistry(Register);
    MiscFlags := IntToStr(FDocMiscStatus);
    // Add reg keys under CLSID
    CreateRegKey(ClassKey+'DocObject', '', MiscFlags);
    CreateRegKey(ClassKey+'Programmable', '', '');
    CreateRegKey(ClassKey+'Insertable', '', '');
    CreateRegKey(ClassKey+'InprocHandler32', '', FHandler);
    // Add reg keys under ProgID
    CreateRegKey(ProgKey+'DocObject', '', MiscFlags);
    CreateRegKey(ProgKey+'Insertable', '', '');
    // Need to remove "control" key added
    // by inherited method
    DeleteRegKey(ClassKey + 'Control');
  end else begin
    DeleteRegKey(ClassKey + 'DefaultExtension');
    DeleteRegKey(ClassKey + 'DefaultIcon');
    DeleteRegKey(ClassKey + 'DocObject');
    DeleteRegKey(ClassKey + 'Programmable');
    DeleteRegKey(ClassKey + 'Insertable');
    DeleteRegKey(ClassKey + 'InprocHandler32');
    DeleteRegKey(ProgKey + 'DocObject');
    DeleteRegKey(ProgKey + 'Insertable');
    inherited UpdateRegistry(Register);
  end;
end;
end.
```

➤ *Above: conclusion of Listing 6*

➤ *Below: Listing 7*

```
unit Main;
interface
uses
  ComObj, ActiveX, AxDocs, Menus, ComCtrls, DAXDoc_TLB;
type
  TDelphiAxDoc = class(TActiveXDocument, IDelphiAxDoc,
    IPersistStreamInit)
  private
    FItem: TMenuItem;
    FSubItem: TMenuItem;
  protected
    { IPersistStreamInit }
    function IPersistStreamInit.Load = PersistStreamLoad;
    function IPersistStreamInit.Save = PersistStreamSave;
    function IsDirty: HResult; stdcall;
    { Override TActiveXControl streaming mechanism
      to simply use memo lines }
    procedure LoadFromStream(const Stream: IStream);
      override;
    procedure SaveToStream(const Stream: IStream); override;
  public
    destructor Destroy; override;
    procedure DoMenuClick(Sender: TObject);
    procedure Initialize; override;
  end;
implementation
uses
  ComServ, StdCtrls, MainForm, AxCtrls, Windows, Dialogs;
destructor TDelphiAxDoc.Destroy;
begin
  inherited Destroy;
  FSubItem.Free;
  FItem.Free;
end;
procedure TDelphiAxDoc.DoMenuClick(Sender: TObject);
begin
  with TColorDialog.Create(nil) do begin
    if Execute then
      (Control as TMemo).Color := Color;
    Free;
  end;
end;
procedure TDelphiAxDoc.Initialize;
begin
  inherited Initialize;
  FItem := NewItem('&Change color', 0, False, True,
    DoMenuClick, 0, 'ColorItem');
  FSubItem := NewSubMenu('&Edit', 0, 'EditItem', [FItem]);
  Menu := NewMenu(Control, 'MainMenu', [FSubItem]);
end;
procedure TDelphiAxDoc.LoadFromStream(
  const Stream: IStream);
var
  OS: TOleStream;
begin
  OS := TOleStream.Create(Stream);
  try
    (Control as TMemo).Lines.SaveToStream(OS);
  finally
    OS.Free;
  end;
end;
procedure TDelphiAxDoc.SaveToStream(const Stream: IStream);
var
  OS: TOleStream;
  Memo: TMemo;
begin
  OS := TOleStream.Create(Stream);
  try
    Memo := Control as TMemo;
    Memo.Lines.LoadFromStream(OS);
    Memo.Modified := False;
  finally
    OS.Free;
  end;
end;
function TDelphiAxDoc.IsDirty: HResult;
begin
  if (Control as TMemo).Modified then
    Result := S_OK
  else
    Result := S_FALSE;
end;
initialization
  TActiveXDocumentFactory.Create(ComServer, TDelphiAxDoc,
    TMemo, Class_DelphiAxDoc, 0, 131473, tmApartment, '',
    8 {DOCMISC_NOFILESUPPORT});
finalization
end.
```

➤ *Listing 8*

```
<HTML>
<HEAD>
<TITLE>Test page for Delphi ActiveX Document</TITLE>
</HEAD>
<BODY>
<center><h1>Test page for Delphi ActiveX
Document</h1></center>
<OBJECT ID="DAXDoc.DelphiAxDoc" <
 CLASSID="CLSID:C10BE34F-A81F-11D2-AF31-0000861EF0BB">
>
</OBJECT>
</BODY>
</HTML>
```