Enhanced VBX Support

by Stephen Posey

One of Delphi's more interesting and impressive features is its support for using Visual Basic custom controls (VBXs) within the Delphi IDE, such that they behave pretty much as if they were 'native' Delphi controls. Delphi admittedly only supports VBXs which conform to the Version 1 specification (a limitation that both Borland C++ and MS-Visual C++ also share, by the way), but this is not as restrictive as it may at first sound. In general, any VBX which doesn't make use of the data aware (database access) features of the VBX specification is (or could be written to be) a Version 1 VBX.

With all the hullabaloo about OCXs and 32-bit 'distributable object' component architectures, it's not clear how much life is actually left in the VBX model; on the other hand, the variety and number of them that are presently available suggests that 16-bit VBXs will be with us for a while yet (it seems unlikely that *everyone* will immediately switch to 32-bit Windows).

VBXs And TVBXControl

Installing a VBX into the Delphi component palette is a pretty straightforward process: simply a matter of picking `Options|Install Components...|VBX` and searching for the .VBX file in question. What may be less clear is what is actually going on when one does this.

When a VBX is installed into Delphi, the installation routine interprets the contents and behavior of the VBX in terms of a specialized VCL component called `TVBXControl`. Any time a VBX is installed in this fashion, the Delphi VBX installation routine generates a *descendent* of `TVBXControl` that acts as a wrapper to make the properties and events which the VBX supports available in appropriate Delphi terms. This component is then compiled into Delphi's COMPLIB.DCL (or a specialized component library if you specify one), just as any normal VCL component would be added to the component palette.

One result of this automaticity on Delphi's part is that the generated `TVBXControl` descendent is very generic, because Delphi is only able to make educated, but nonetheless somewhat limited, guesses with regard to the VBX's actual capabilities.

As an example, Listing 1 shows the automatically generated `TVBXControl` for the BiSwitch VBX that comes with Delphi. BiSwitch is a relatively simple VBX, but, as you can see Delphi nonetheless generates a great deal of code for it. Because documentation of `TVBXControl`'s operation is somewhat sparse, I'm still learning about the more arcane aspects of all this (like just how the 'default init data' is used, and how the event passing mechanics work). However, the really interesting part for our purposes is the `published` section of `TBiSwitch`. The `published` section of a VCL component is what provides the design time interface to a VCL control; whichever properties are defined here are the ones which will appear in Delphi's Object Inspector when the component is added to a project.

The Delphi on-line *Component Writer's Help* has this to say about the `TVBXControl` Component: *"Unlike other component types, `TVBXControl` is not available for you to create direct descendents. If you want to create a Delphi component that derives from a VBX control, first install the VBX control into the Component palette as described in the Delphi User's Guide. Then, using the generated component as a starting point, you can add your own properties and methods to the VBX component."*

I would take this a step further and assert that the automatically generated VBX component need not necessarily be installed in the component palette at all (at least not permanently, I confess that I haven't found a way to get Delphi to generate the `TVBXControl` descendent wrapper code without first installing the VBX, I suspect that there may be no such critter). The main problem with the automatically generated wrappers is that the properties that are published tend to have a cruder, or more Visual Basic-like, 'feel' *[Note 1]* to them, or just don't fit well into the Delphi component 'idiom'.

I think that, except for the simplest VBXs, the automatically generated VBX wrapper should probably best be treated like the `TCustom...` components provided by the VCL, ie as ancestors for your own specialized descendent components. With this in mind, the automatically generated class should probably not actually publish *anything*. To bring this about, it's necessary to edit the automatically generated code that Delphi produces *[Note 2]*.

To change the BiSwitch code in Listing 1 to turn it into an abstract ancestor component simply move the `published` keyword to the end of the list of properties, so that (for this example anyway) *none* of the properties which are specific to `TBiCustomSwitch` would appear in the Object Inspector if it were installed onto the component palette. You could also rename the component, as I have done in the file SWITCH2.PAS on the disk. If you don't rename the component, make sure that you *never* try to install two components with the same name *[Note 3]*).

Descending from this abstract component gives the programmer full control over how the component will actually appear at design time. If you do install this component, you will find that, despite moving the `published` keyword, some properties still *do* appear in the Object Inspector: these are

properties that TVBXControl itself is inheriting from its ancestor TComponent. Since TVBXControl descends from TComponent, it also has access to some generic Delphi component properties that the default VBX interpretation does not include by default; some of these can be used to make VBXs more Delphi-like in their design time and run-time behavior, which is really the point of all this.

Customising ChartFX

To create a more illustrative (and complex) example than BiSwitch, I picked the ChartFX component (which everyone who owns Delphi will also have available). I took Delphi's default wrapper, moved the published keyword as described above, renamed TChart to TCustChart and also renamed the unit to CUSTCHRT.PAS [*Note 4*].

Next, I created a new component using File | New Component... called Chart2FXEx which descends from TCustChart, and placed it on the VBX tab of the component palette, saving the new component unit as CHRTFXEX.PAS.

Finally, within the published section of Chart2FXEx I then began to reveal parts of TCustChart (and TComponent) that made Chart2FXEx behave more like a native Delphi component.

In general, there are four types of properties which you will encounter in this process. Firstly, those that you'll publish 'as-is', secondly those inherited from elsewhere in the VCL, which aren't automatically added to VBXs, thirdly those for which a Delphi property exists that is equivalent to a VBX property, but the Delphi property can be used to make the VBX seem more native, and lastly those that can be handled by a VCL component mechanism in a more Delphi-like fashion than the automatically generated one *[Note 5]*.

Listing 2 shows one possible set of custom properties that can be added to Chart2FXEx, some of which fall into each of the categories mentioned above:

1). Unless you're planning to do some fancy footwork with the VBX's default event handling, it's probably best to go ahead and publish all of the OnEvent handler properties (this provided a minor snag in publishing one of the VCL descendent properties which I'll discuss below). Similarly, any of the dialog based properties (AdmDlg, FontDlg, CustTool, pType) should be passed through (unless you're into creating your own property editors for these). Most Boolean or numeric based properties (such as Autoincrement, BottomGap, Decimals...) can also just be passed on through. It's really a

➤ *Listing 1*

```
unit Switch;
interface
uses
  SysUtils, Classes, Graphics, Forms, Controls,
  VBXCtrl, BIVBX;
{ $DEFINE InitTBiSwitch}
{^- Remove space to enable default init data -
    See documentation. }
type
  TBiSwitchOnEvent = procedure (Sender: TObject) of object;
  TBiSwitchOffEvent = procedure (Sender: TObject) of object;
  { TBiSwitch }
  TBiSwitch = class(TVBXControl)
  protected
    FOnOn: TBiSwitchOnEvent;
    FOnOff: TBiSwitchOffEvent;
    { Event handler }
    procedure HandleVBXEvent(var Message: TWMVBXFireEvent);
      override;
  public
    constructor Create(AOwner: TComponent); override;
    property Index: TVBInteger index 1 read GetIntProp;
  published
    property Visible;
    property ForeColor: TColor index 10 read GetColorProp
      write SetColorProp default -2147483640;
    property BackColor: TColor index 11 read GetColorProp
      write SetColorProp default -2147483633;
    property Caption;
    property BorderStyle: TVBEnum index 13
      read GetEnumProp write SetEnumProp default 0;
    property pOn: Boolean index 14 read GetBoolProp
      write SetBoolProp;
    property TextPosition: TVBEnum index 15
      read GetEnumProp write SetEnumProp default 0;
    property Font;
    property ParentFont;
    property TabStop;
    property TabOrder;
    property DragMode;
    property DragCursor;
    property OnOn: TBiSwitchOnEvent read FOnOn write FOnOn;
    property OnOff: TBiSwitchOffEvent read FOnOff
      write FOnOff;
    property OnMouseMove;
    property OnMouseDown;
    property OnMouseUp;
    property OnKeyDown;
    property OnKeyUp;
    property OnKeyPress;
    property OnEnter;
    property OnExit;
    property OnDragOver;
    property OnDragDrop;
    property OnEndDrag;
  end;

procedure Register;
implementation
{ Default form data for TBiSwitch }
{$IFDEF InitTBiSwitch}
const
  TBiSwitchInitLen = 57;
procedure TBiSwitchInitData; near; assembler;
asm
  DB $00,$08,$42,$69,$53,$77,$69,$74,$63,$68,$01,$00,$00,$02,$06,$FF
  DB $FF,$08,$00,$0A,$00,$00,$00,$00,$0B,$0F,$00,$00,$80,$0C,$08,$42
  DB $49,$53,$57,$49,$54,$43,$48,$0D,$00,$0E,$00,$00,$0F,$00,$10,$00
  DB $16,$00,$00,$17,$00,$00,$18,$00,$FF
end;
{$ENDIF}
{ TBiSwitch }
constructor TBiSwitch.Create(AOwner: TComponent);
begin
  FVBXFile := StrNew('SWITCH.VBX');
  FVBXClass := StrNew('BISWITCH');
  SetBounds(0, 0, 80, 32);
  inherited Create(AOwner);
  ControlStyle :=
    ControlStyle - [csCaptureMouse, csClickEvents];
  TabStop := True;
  FVBXFlags := [vfProcessMnemonic];
  {$IFDEF InitTBiSwitch}
  FHForm := VBXCreateFormFile(TBiSwitchInitLen,
    @TBiSwitchInitData);
  {$ENDIF}
end;
procedure TBiSwitch.HandleVBXEvent(var Message:
  TWMVBXFireEvent);
begin
  case Message.VBXEvent^.EventIndex of
    0: DispatchCustomEvent(FOnOn, Message, 0);
    1: DispatchCustomEvent(FOnOff, Message, 1);
    2: DispatchMouseMoveEvent(OnMouseMove, Message);
    3: DispatchMouseEvent(OnMouseDown, Message);
    4: DispatchMouseEvent(OnMouseUp, Message);
    5: DispatchKeyEvent(OnKeyDown, Message);
    6: DispatchKeyEvent(OnKeyUp, Message);
    7: DispatchKeyPressedEvent(OnKeyPress, Message);
    8: begin end; { VCL Handles OnGotFocus }
    9: begin end; { VCL Handles OnLostFocus }
    10: begin end; { VCL Handles OnDragOver }
    11: begin end; { VCL Handles OnDragDrop }
  end;
end;
{ Designer registration }
procedure Register;
begin
  RegisterComponents('VBX', [TBiSwitch]);
end;
end.
```

```
unit ChrtFXEx;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics, Controls, Forms, Dialogs, Menus,
  ChartFX, {import unit for ChartFX constants and function prototypes}
  CustChrt; {abstract ChartFX VBX wrapper unit }
type
  tcfxGridType =
    (cfxNoGrid, cfxHorzGrid, cfxVertGrid, cfxBothGrid);
type
  TChart2FXEx = class(TCustChart)
    private
      function GetBorderStyle : TBorderStyle;
      procedure SetBorderStyle(Value : TBorderStyle);
      function GetGrid : tcfxGridType;
      procedure SetGrid(Value : tcfxGridType);
    protected
    public
    published
    property BorderStyle: TBorderStyle
      read GetBorderStyle write SetBorderStyle
      default bsSingle;
    property Grid : tcfxGridType read GetGrid
      write SetGrid default cfxNoGrid;
    property Align;
    property Hint;
    property PopupMenu;
    property ParentShowHint;
    property TabOrder;
    property TabStop;
    property OnClick;
    property OnMouseDown;
    property OnMouseUp;
    property OnMouseMove;
    property OnRButtonDown;
  end;
procedure Register;
implementation
```

```
function TChart2FXEx.GetBorderStyle : TBorderStyle;
begin
  case inherited BorderStyle of
    0 : Result := bsNone;
    1 : Result := bsSingle;
  end (* case *);
end;
procedure TChart2FXEx.SetBorderStyle(Value :
  TBorderStyle);
begin
  case Value of
    bsNone   : inherited BorderStyle := 0;
    bsSingle : inherited BorderStyle := 1;
  end (* case *);
end;
function TChart2FXEx.GetGrid : TcfxGridType;
begin
  case inherited Grid of
    CHART_NOGRID   : Result := cfxNoGrid;
    CHART_HORZGRID : Result := cfxHorzGrid;
    CHART_VERTGRID : Result := cfxVertGrid;
    CHART_BOTHGRID : Result := cfxBothGrid;
  end (* case *);
end;
procedure TChart2FXEx.SetGrid(Value : TcfxGridType);
begin
  case Value of
    cfxNoGrid   : inherited Grid := CHART_NOGRID;
    cfxHorzGrid : inherited Grid := CHART_HORZGRID;
    cfxVertGrid : inherited Grid := CHART_VERTGRID;
    cfxBothGrid : inherited Grid := CHART_BOTHGRID;
  end (* case *);
end;
procedure Register;
begin
  RegisterComponents('VBX', [TChart2FXEx]);
end;
initialization
end.
```

➤ *Listing 2*

judgment call whether or not these should actually have been left published in `TCustChart`, or should be published here; my opinion on this is that forcing *all* properties to have to be explicitly published here makes `TCustChart` most flexible.

2). There are a number of VCL properties which can make using VBXs a much more pleasant experience:

- `Align`: One of Delphi's most useful component properties – publishing this one property alone can save you a whole slew of 'fiddly' code to get your VBX to re-size itself with the form window. The Delphi VBX interpreter routines can't tell from the VBX itself whether or not this property is appropriate for a given VBX, so you have to add it yourself *[Note 6]*.
- `PopupMenu`: I've successfully managed to get popup menus to work with a number of other VBXs that I performed this type of work on, but couldn't seem to get it to work with `ChartFX`. I then realized that this VBX exports a built-in `OnRButtonDown` event handler which was probably overriding the VCL handler for that event. I got around this by simply adding code to the event handler to call up the popup menu when the right mouse button was clicked.
- Others that may be appropriate to add are `ParentShowHint` and `ParentColor`

3). Delphi's VCL already includes a property for `BorderStyle` that has equivalent functionality to the `BorderStyle` property ported from the VBX, but it looks different from the VCL property at design time, and requires use of 'magic numbers' to change it at run-time. Publishing a 'wrapper' property and `read` and `write` methods (as in Listing 1) makes `BorderStyle` look like a true Delphi property. There will probably be few of this kind of property in your VBX porting efforts. An interesting exercise might be to find a way to use Delphi's font property editor as an alternative to the ChartFX `FontDlg`.

4). The `CHARTFX.INT` file (which should be in your \DELPHI\DOC directory) provides a large set of numeric constants that serve to delineate various properties of ChartFX. In order to integrate this information into my custom components more easily, I turned CHARTFX.INT into an import unit (all I had to do was add `end.` after the `implementation` keyword, save the file as CHARTFX.PAS and compile it), which I could add to the `uses` clause of my descendent component.

One of the easiest properties to handle is the `Grid` property, which only has four possible states. To make this more Delphi-like, I created a custom enumerated type (`tcfxGridType`) which equated to each of the `Grid` type values. I published an over-riding property of that type with associated custom `read` and `write` methods. Other properties of ChartFX which could be similarly handled are `ChartType`, `LineStyle` and `PointType`.

After you've added or modified the properties you want your VBX to have, you can add this new component to the Component Palette just like you would any *native* VCL component (ie add the `.PAS` file you just created, *not* the VBX). I hope this gives you an idea of how it's possible to treat VBXs in pretty much the same way as a native VCL component. Of course, if you distribute an application that uses a VBX you still have to include the `.VBX` file itself, and any ancillary files that the VBX requires (check your VBX's documentation to find this out).

The ChartFX VBX is a large and complex component, with many opportunities for customization. I'm similarly investigating the VBXs which come with Borland's Visual Solutions Pack to integrate them better into Delphi. If there's interest, I can publish my findings in a future piece.

Stephen L Posey works in New Orleans, USA, he can be contacted by email at SLP@uno.edu

Notes to the article:

1. A somewhat nebulous attribution, I know; it's really an aesthetic issue, but then, so much of using (and programming for!) a GUI is aesthetic anyway.

2. One feature that I'd really like to see in future versions of Delphi is the ability to customize the code that Delphi generates automatically. While my coding style is pretty similar to 'Borland Standard', it would still be nice to be able to make some minor aesthetic or preferential modifications. In particular, being able to modify how the default VBX code is generated would save a couple of awkward steps in the procedure I'm describing in this article.

3. Delphi's response to that is *very* (and remarkably) unfriendly: it ends up giving you a completely empty component palette. The original has to be rebuilt from scratch. To protect yourself from this eventuality (and any others that might clobber the component palette), it's wise to make a backup copy of `COMPLIB.DCL` before doing any serious work with custom components. Also, since the abstract component isn't meant to be installed into Delphi's Component Palette anyway, the references to `RegisterComponents()` have become extraneous and can be removed – leaving them there won't hurt anything, however.

4. I then had to compile `CUSTCHRT.PAS` to create a `.DCU` file. This can be done a couple of different ways: either temporarily install `TCustChart` into the Component Palette, which causes Delphi to compile it as part of `COMPLIB.DCL`, or (and this is my preference) add `CUSTCHRT.PAS` to a 'dummy' project and do a `Compile|Build All`.

5. There is, of course, a fifth category: custom properties which you add yourself, but this is no different really from adding custom properties to any VCL component, and are beyond the scope of this article.

6. Actually, wanting `Align` for another VBX was the original impetus for my investigating `TVBXControl`'s ability to inherit from `TControl` in the first place, which then led to all of this.