VISUAL DEVELOPER Delphi by Design

One-Step ActiveX - Part 2

by Ray Konopka
April/May 1998, Vol. 9, No. 1 -- Download Source Code: **Click Here**

It takes only one step to make a working ActiveX con a VCL control—but there may be some additional fuss VCL-style design-time features like property editors.

Way back in the August/September 1997 issue, we began a three-part (Component Object Model) support in Delphi. Throughout the series, we sav interface type in Delphi 3 plays an important role in supporting Micro particular, we discovered how interfaces are used to manipulate objects, def of automation servers, and implement Windows shell extensions.

In the last issue, all of the information from the previous three article together to describe the process of converting native Delphi component controls. Delphi 3 provides a custom wizard that makes the initial conve easy. Just tell the wizard which component you would like to convert, and t the rest. The process is so simple that it's commonly referred to as "One-

Attaining one-step Ac-tiveX conversion in practice, however, requires forethought. As we saw in the last issue, the ActiveX wizard does a good jo *functional* ActiveX control from a Delphi component. However, because differences between the Visual Component Library (VCL) and the A framework, creating an ActiveX control that is *functionally equivalent* component is not often possible. You can help the process along by followin that were introduced in the last issue, but these are not foolproof. The poi using the wizard to convert a component, it is often necessary to modif ActiveX control manually.

In this issue, we'll continue with the ListBoxX example that was introdu column. However, this time, instead of focusing on runtime improvements we're going to focus on enhancing the control's design-time features. In discover how to provide the equivalent of property editors and component ActiveX control (for example, drop-down list property editors and pr Wrapping up this article is a discussion of distribution and regis

Property Lists

Consider the ListBoxX control shown in Figure 1. An instance of ListBoxX ha onto a Visual Basic form. However, notice the Cursor and DragCursor pr Property Inspector. Instead of the usual crDefault and crDrag values, we The TCursor type in Delphi is just an integer range, so when the ActiveX converted the TListBox component, the cursor properties were treated j integer properties. As a result, to change the cursor of the list box to an must set the Cursor property to -11, which is the value of the crHour defined in the Controls unit. Not very intuitive, is it?

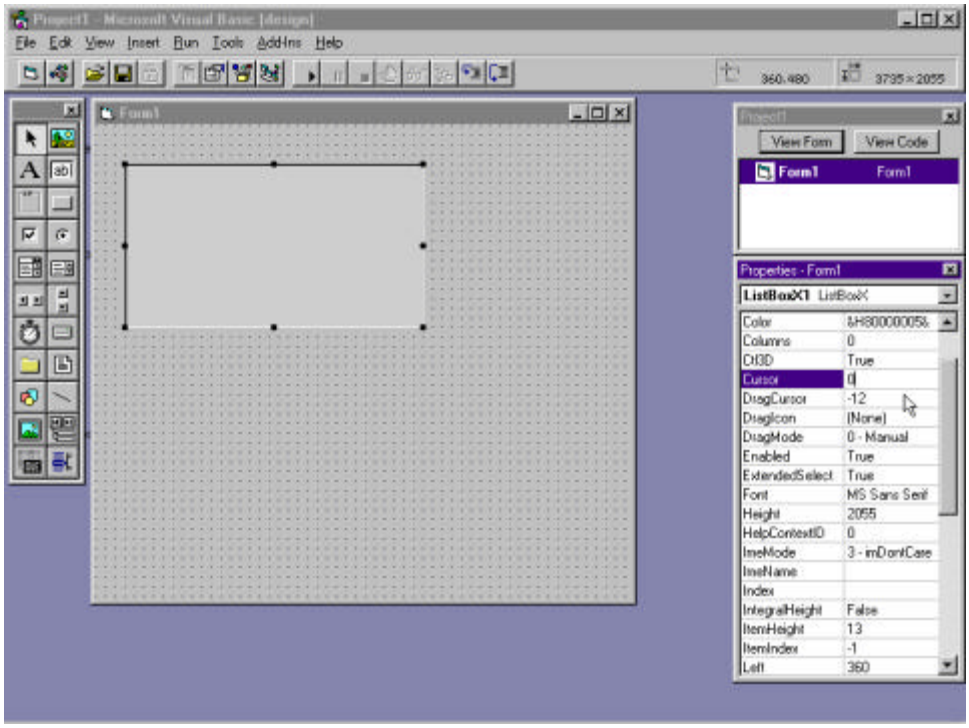

Figure 1: Changing cursors the hard way.

When using the TListBox component in Delphi, in addition to displaying a instead of a number, the Object Inspector allows the user to display a drop possible cursor values. Of course, the items in the list are symbolic names as it be great if we could do the same thing in our ActiveX control? Well, we can the GetPropertyString, GetPropertyStrings, and GetPropertyValue metho defined in the TActiveXControl class.

Listing 1 shows a partial listing of the ListBoxImpl unit. Recall from last tim an ActiveX control in Delphi involves creating a COM wrapper around a component. In our example, TListBoxX is the COM wrapper, defined as a TActiveXControl. As a result, we can override the above methods within class.

GetPropertyString is called whenever a property inspector displays a propert this method, we are given the opportunity to provide a string representation value. If you've written a property editor for a Delphi component before, y similarity between GetPropertyString and TPropertyEditor.GetValue.

GetPropertyString is passed two parameters. The first is the dispid of the p string representation is being requested. Because the same method is properties of the control, we must filter out only the properties we want to easily accomplished by writing a case statement on the dispid passed to t determine which dispid maps to a particular property, look in the type library For example, **Listing 2** shows a partial listing for the DelphiByDesignXLib_T interface unit. The mappings can be found in the IListBoxXDis

The second parameter to GetPropertyString is a variable string paran parameter to the string value to be displayed as the property value. In add set the return value of the method to True. Returning False causes the pro to ignore the string parameter.

The GetPropertyString method in **Listing 1** uses the Delphi CursorToStri convert the current property value into a symbolic name. Notice how the G Get_DragCursor methods are used to access the current pro

One more note with respect to GetPropertyString: There is a bug in the

Delphi that prevents the GetPropertyString method from being called. T
caused by an incomplete TActiveXControl.GetDisplayString method. Fo
problem has been corrected in the 3.02 release of Delphi, which, by the
recommend getting. The improvements to the help files and examples alone
time that it takes to download and apply the patch from the Borland Web s
these instructions assume that you have already upgraded to 3.01.

Okay, back to work. Implementing the GetPropertyString method will ensure
name appears in the property inspector for both cursor properties. But w
drop-down list of all possible cursors? Both the GetPropertyStrings and Ge
must be overridden to support a drop-down list. GetPropertyStrings is
populating a string list with the values that will be displayed in the drop-d
the user selects one of the items from the list, the GetPropertyValue meth
retrieve the actual property value. Although it sounds simple enough, th
subtleties that must be considered when overriding these t

Let's look at GetPropertyStrings first. This method takes two parameters:
string list. The dispid again identifies the property being manipulated, while
holds the values to be displayed in the drop-down list. In addition, the strin
store a unique "cookie" value for each item in the list. When the user select
the list, the corresponding cookie value is passed to the GetProperty

To associate a cookie value with a particular string value in the list, add th
list by calling the Strings.AddObject method. In the string parameter, pass th
appear in the list for the entry. In the object parameter, pass a unique cook
can be anything you choose. In our example, the cookie used in the cursor li
value itself. Note that you may have to typecast the cookie into a TObject
call the AddObject method.

When the user selects an item from the drop-down list, the GetPropertyV
called and is passed three parameters. The first is our good friend the
property. The second is the cookie value corresponding to the item chosen.
a variant that represents the property value. The processing of this method
For the selected property, set the variant to the property value corresp
cookie value passed in the second parameter. In our example, we just ass
value to the variant.

Once these three methods are defined, all we need to do is rebuild the libra
out the control. Figure 2 shows a more user-friendly property inspector fo
control, at least in terms of the cursor properties.

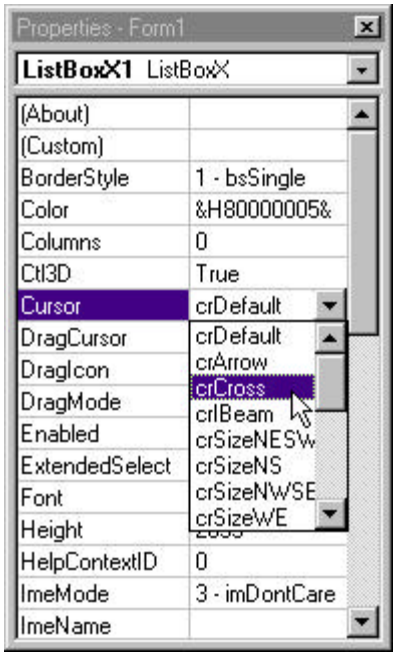

Figure 2: Selecting a new cursor from a list.

Predefined Property Pages

Take a closer look at the property inspector in Figure 1. Notice that the It absent from the list. Where is it? The ListBoxX control does have an Item simply not published. Therefore, the only way to populate the ListBoxX contr the Items property at runtime. Fortunately, Delphi provides a way to acc property at design-time by using a predefined property page.

Property pages are very similar to Delphi component editors in that you can page to edit multiple properties of an ActiveX control. Property pages are property editors in that you can have multiple property pages per ActiveX provides four predefined property pages that you can associate with your Ac Each page has a corresponding Class ID, which is declared in the AxCtrls un each ID and describes the functionality of each page.

TABLE 1

Class IDs for Delphi predefined property pages.

Class ID	Description
Class_DColor PropPage	Property page for editing all **TColor** properties in a control.
Class_DFont PropPage	Property page for editing all **TFont** properties in a control.
Class_DPicture PropPage	Property page for editing all **TPicture** properties in a control.
Class_DString PropPage	Property page for editing all **TStrings** properties in a control.

These predefined property pages are designed in such a way that they ca any ActiveX control. Each property page uses runtime type information (RTT which properties in the control can be edited using the property page and t property name to a combo box. This way, a single property page can edit properties that fall into one of these four types. Figure 3 shows the Strings being used to edit the Items property of a ListBoxX control.

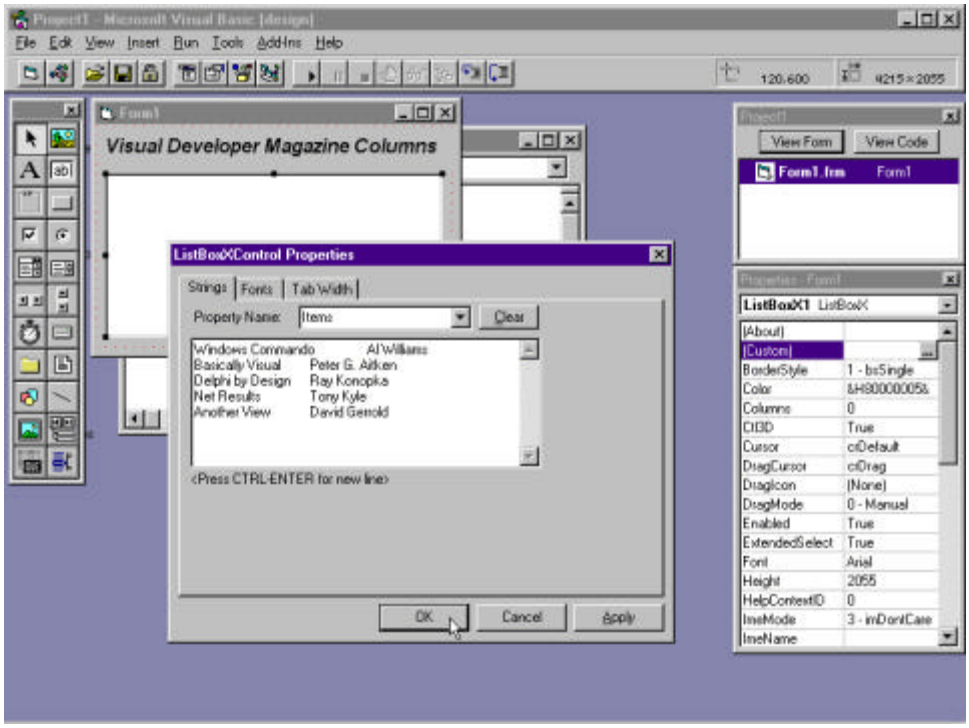

Figure 3: Using a predefined property page.

To associate a predefined property page with your ActiveX control, simply DefinePropertyPage in your control's DefinePropertyPages method. For exar shows how the string and font property pages are associated with the Li

Custom Property Pages

In addition to using predefined property pages, you can construct you property pages to be used on your ActiveX control. To create a new proper File|New, switch to the ActiveX page, and select the Property Page item. D a form file and unit that represent the property page. **Listing 3** shows the s the TabWidthPpg property page unit. Notice that TPpgTabWidth de TPropertyPage, which in turn descends from TCustomForm. Therefore, y property page just like a normal Delphi form.

For the ListBoxX control, I decided to create a property page that allows u adjust the TabWidth property. Figure 4 shows the property page in action.

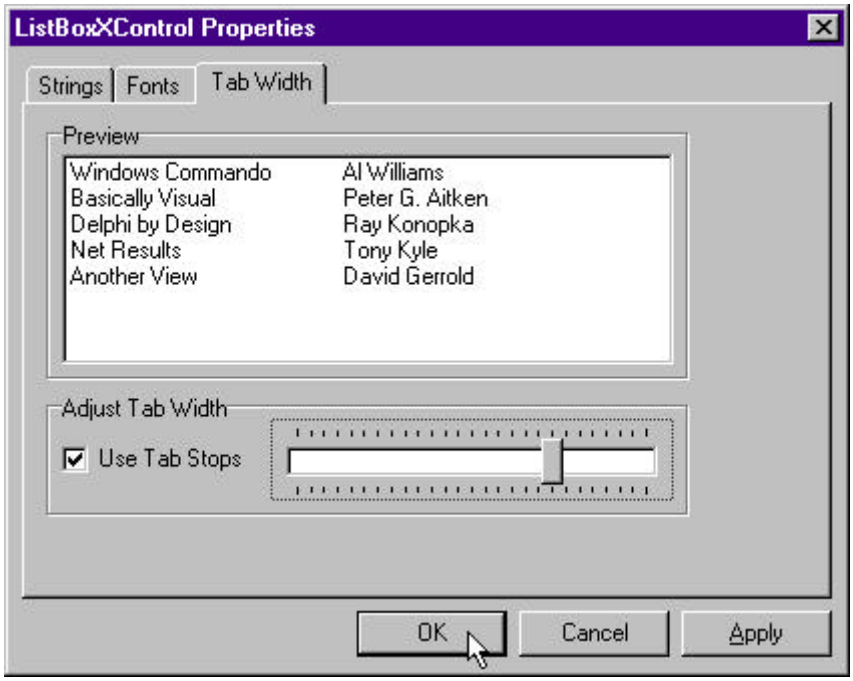

Figure 4: The custom TabWidth property page.

As mentioned earlier, the notion of a property page is very similar to that of property editor or a component editor in Delphi. However, there are some between the two. First, you don't have to worry about placing OK and Can the page. Because multiple property pages may exist for a control, all prop displayed on separate tabs within a PageControl. The environment invokin pages is responsible for supplying the OK, Cancel, and Apply buttons.

Second, the caption of a property page form supplies the text used environment to identify the page. Notice in Figure 4 that the current tab Width".

Next, you need to add code to the property page unit that will update the the property page form with the data stored in the ActiveX UpdatePropertyPage method is automatically defined for just this p TPpgTabWidth.UpdatePropertyPage method, which appears in **Listing 3**, LstPreview list box with the strings stored in the ActiveX control. Next, the and the TabWidth property are initialized.

Once the property page is invoked, we must be able to update the Active the new values selected in the property page. This is accomplished by UpdateObject method, which is also automatically generated by TPpgTabWidth.UpdateObject method simply sets the TabWidth property control to the TabWidth setting of the preview list.

It is important to note that OleObject represents the ActiveX contro underlying Delphi component. Therefore, the methods and properties d control must be used to transfer data.

Before a property page can be invoked on an ActiveX control, you must property page with the control. This is accomplished by adding a DefinePropertyPages method in the wrapper class for your ActiveX con property page that you want to associate with the control, you ma DefinePropertyPage and pass it the class ID of the property page. Delph generates a GUID for new property pages and assigns it to a constant that to DefinePropertyPage . The constant is declared in the property pa DefinePropertyPages method in **Listing 1** associates the PpgTabWidth prop the ActiveX control. Don't forget to add the property page unit to the uses

implementation unit.

One final word regarding property pages: If you add a custom property page server that has already been registered, the new property page will not be you re-register the ActiveX server. Simply compiling the library to generate a is *not* sufficient. You must register the server again so that Windows become property page.

Distributing ActiveX Controls

There are a couple of issues involved in distributing ActiveX controls. First you must distribute the *.ocx file that contains your ActiveX controls. If build your ActiveX library project using runtime packages, you will also need of the runtime packages required by your component.

If you instruct the ActiveX control wizard to generate a design-time li ActiveX control, you will also need to distribute the correspond

If you deploy an ActiveX control library that uses the IStrings interface or font, color, strings, or picture property pages, then you must also deploy th type library, which comes in two forms: the StdVcl32.dll library, and a s library called StdVcl32.tlb. Both files are located in the Windows System example, C:\WinNT\System32) after Delphi 3 is installed.

Control libraries that use the predefined property pages must deploy StdVcl3 if the ActiveX controls only use the IStrings interface, the StdVcl32.tlb type deployed instead. Regardless of which file is deployed, the file must be re system registry, just like the ActiveX control library.

The Turbo Register Server Utility

The system registry can be updated with the information needed to suppo control by using the Turbo Register Server (TRegSvr) utility provided as a d in Delphi. The project is located in the Demos\ActiveX\TRegSvr directory be Delphi 3 installation directory. Of course, before you'll be able to use the need to compile it.

TRegSvr has a simple command-line interface. After the program name, yo of options followed by the file to register. The file can be an ActiveX serve library. The default action is to register the specified file. The -u opti unregister the specified file. The -t option is used to indicate that t contained in the specified file should be registered. This option is not needed file ends in ".tlb". The -q option instructs the TRegSvr program to operate by not displaying any output. This option makes TRegSvr ideal to installation programs.

For the examples presented earlier, the following calls to TRegSvr handle TListBoxX ActiveX control with the system:

```
TRegSvr -q DelphiByDesignXLib.ocx
TRegSvr -q StdVcl32.dll
```

On the Drawing Board

Next time, *Visual Developer Magazine* celebrates its 50th issue. Ever sin Pascal" column (the precursor to "Delphi by Design") first appeared in *PC Te* ago, I have received quite a bit of feedback. Some of the messages were sin Others asked where the source code from the column could be found. But t common questions asked were about Delphi programming techniques. Tha questions.

For example, the following is a typical message: “Hi, Ray. I read your colur Your article on <some topic> was excellent. However, I’ve run into a progra that I’m hoping you’ll be able to help me with. How do I <some task> in De maybe I exaggerated the tone of the first two sentences.

The point is that I receive a lot of good questions, and because many of the keep coming up, I have often wanted to incorporate a topic or two Unfortunately, I never have any extra room in my column. Therefore, the ne› “Delphi by Design” will not have a single topic. Instead, I’m putting together topics based on questions that I have received. v

---

Listing 1 - ListBoxImpl.src

```
unit ListBoxImpl;

interface

uses
  Windows, ActiveX, Classes, Controls, Graphics, Menus, Forms, StdCtrls,
  ComServ, StdVCL, AXCtrls, DelphiByDesignXLib_TLB;

type
  TListBoxX = class( TActiveXControl, IListBoxX )
  private
    { Private declarations }
    FDelphiControl: TListBox;
    . . .
  protected
    { Protected declarations }
    procedure InitializeControl; override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;

    procedure DefinePropertyPages( DefinePropertyPage:
                              TDefinePropertyPage); override;

    function GetPropertyString( DispID: Integer;
                           var S: string ): Boolean; override;
    function GetPropertyStrings( DispID: Integer;
                            Strings: TStrings ): Boolean; override;
    procedure GetPropertyValue( DispID, Cookie: Integer;
                           var Value: OleVariant ); override;

    { Methods that support properties }
    . . .
  end;

implementation

uses
  TabWidthPpg, AboutListBox, SysUtils;


{ TListBoxX }


procedure TListBoxX.DefinePropertyPages(
```

```
  DefinePropertyPage: TDefinePropertyPage );
begin
  { Associate Predefined Property Pages with this control }
  DefinePropertyPage( Class_DStringPropPage );
  DefinePropertyPage( Class_DFontPropPage );

  { Associate a Custom Property Page with this control }
  DefinePropertyPage( Class_PpgTabWidth );
end;


function TListBoxX.GetPropertyString( DispID: Integer;
                                var S: string ): Boolean;
begin
  case DispID of
    5:    //  5 = DispID for DragCursor property in IListBoxXDisp
    begin
      S := CursorToString( Get_DragCursor );
      Result := True;
    end;

    26:   // 26 = DispID for Cursor property in IListBoxXDisp
    begin
      S := CursorToString( Get_Cursor );
      Result := True;
    end;

    else
      Result := False;
  end;
end;


function TListBoxX.GetPropertyStrings( DispID: Integer;
                              Strings: TStrings ): Boolean;
var
  I: Integer;
  Cookie: Integer;
  TempList: TStringList;
begin
  case DispID of
    5,    //  5 = DispID for DragCursor property in IListBoxXDisp
    26:   // 26 = DispID for Cursor property in IListBoxXDisp
    begin
      TempList := TStringList.Create;
      try
        GetCursorValues( TempList.Append );
        for I := 0 to TempList.Count - 1 do
        begin
          Cookie := StringToCursor( TempList[ I ] );
          Strings.AddObject( TempList[ I ], TObject( Cookie ) );
        end;
      finally
        TempList.Free;
      end;
      Result := True;
    end;
    else
      Result := False;
  end;
end;
```

```
procedure TListBoxX.GetPropertyValue( DispID, Cookie: Integer;
                              var Value: OleVariant );
begin
  case DispID of
    5,    // 5 = DispID for DragCursor property in IListBoxXDisp
    26:   // 26 = DispID for Cursor property in IListBoxXDisp
    begin
      { Cookie represents the item that was selected }
      Value := Cookie;
    end;
  end;
end;


{= All other support methods deleted for space =}


initialization
  TActiveXControlFactory.Create( ComServer, TListBoxX, TListBox,
    Class_ListBoxX, 1, '{B19A64E4-644D-11D1-AE4B-444553540000}', 0);

end.
```

Listing 2 - DelphiByDesignXLib_TLB.src

```
unit DelphiByDesignXLib_TLB;

{ This file contains pascal declarations imported from a type library.
  This file will be written during each import or refresh of the type
  library editor.  Changes to this file will be discarded during the
  refresh process. }

{ DelphiByDesignXLib Library }
{ Version 1.0 }

interface

uses Windows, ActiveX, Classes, Graphics, OleCtrls, StdVCL;

const
  LIBID_DelphiByDesignXLib: TGUID =
    '{B19A64DB-644D-11D1-AE4B-444553540000}';

const
{ Component class GUIDs }
  Class_ListBoxX: TGUID = '{B19A64DE-644D-11D1-AE4B-444553540000}';

type
{ Forward declarations: Interfaces }
  IListBoxX = interface;
  IListBoxXDisp = dispinterface;
  IListBoxXEvents = dispinterface;

{ Forward declarations: CoClasses }
  ListBoxX = IListBoxX;

{ Forward declarations: Enums }
  TxBorderStyle = TOleEnum;
  TxDragMode = TOleEnum;
  TxImeMode = TOleEnum;
```

```
  TxListBoxStyle = TOleEnum;
  TxMouseButton = TOleEnum;

{ Dispatch interface for ListBoxX Control }

  IListBoxX = interface(IDispatch)
    ['{B19A64DC-644D-11D1-AE4B-444553540000}']
    function Get_DragCursor: Smallint; safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    . . .
    function Get_Cursor: Smallint; safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    . . .
    procedure AboutBox; safecall;
    . . .
    property DragCursor: Smallint
      read Get_DragCursor write Set_DragCursor;
    . . .
    property Cursor: Smallint read Get_Cursor write Set_Cursor;
  end;

{ DispInterface declaration for Dual Interface IListBoxX }

  IListBoxXDisp = dispinterface
    ['{B19A64DC-644D-11D1-AE4B-444553540000}']
    property BorderStyle: TxBorderStyle dispid 1;
    property Color: TColor dispid 2;
    property Columns: Integer dispid 3;
    property Ctl3D: WordBool dispid 4;
    property DragCursor: Smallint dispid 5;
    property DragMode: TxDragMode dispid 6;
    property Enabled: WordBool dispid 7;
    property ExtendedSelect: WordBool dispid 8;
    property Font: Font dispid 9;
    property ImeMode: TxImeMode dispid 10;
    property ImeName: WideString dispid 11;
    property IntegralHeight: WordBool dispid 12;
    property ItemHeight: Integer dispid 13;
    property Items: IStrings dispid 14;
    property MultiSelect: WordBool dispid 15;
    property ParentColor: WordBool dispid 16;
    property ParentCtl3D: WordBool dispid 17;
    property Sorted: WordBool dispid 18;
    property Style: TxListBoxStyle dispid 19;
    property TabWidth: Integer dispid 20;
    property Visible: WordBool dispid 21;
    procedure Clear; dispid 22;
    property ItemIndex: Integer dispid 23;
    property SelCount: Integer readonly dispid 24;
    property TopIndex: Integer dispid 25;
    property Cursor: Smallint dispid 26;
    procedure AboutBox; dispid -552;
  end;

{ Events interface for ListBoxX Control }

  IListBoxXEvents = dispinterface
    ['{B19A64DD-644D-11D1-AE4B-444553540000}']
    procedure OnClick; dispid 1;
    procedure OnDblClick; dispid 2;
    procedure OnKeyPress(var Key: Smallint); dispid 3;
    procedure OnColorItem(Index: Integer;
```

```
                    var Color: TColor); dispid 4;
  end;

implementation

end.
```

Listing 3 - TabWidthPpg.pas

```
unit TabWidthPpg;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  StdCtrls, ExtCtrls, Forms, ComServ, ComObj, StdVcl, AxCtrls,
  ComCtrls;

type
  TPpgTabWidth = class( TPropertyPage )
    GrpPreview: TGroupBox;
    GrpTabWidth: TGroupBox;
    LstPreview: TListBox;
    ChkUseTabs: TCheckBox;
    TrkTabWidth: TTrackBar;
    procedure ChkUseTabsClick(Sender: TObject);
    procedure TrkTabWidthChange(Sender: TObject);
  private
    { Private declarations }
  protected
    procedure UpdatePropertyPage; override;
    procedure UpdateObject; override;
  public
    { Public declarations }
  end;

const
  Class_PpgTabWidth: TGUID =
    '{8BE91420-9070-11D1-AE4B-44455354616F}';

implementation

{$R *.DFM}

procedure TPpgTabWidth.UpdatePropertyPage;
var
  I: Integer;
begin
  { Update your controls from OleObject }

  { Copy strings from control into preview list box }
  for I := 0 to OleObject.Items.Count - 1 do
    LstPreview.Items.Add( OleObject.Items[ I ] );

  ChkUseTabs.Checked := OleObject.TabWidth > 0;
  TrkTabWidth.Position := OleObject.TabWidth div 4;
  LstPreview.TabWidth := OleObject.TabWidth;
end;


procedure TPpgTabWidth.UpdateObject;
```

```
begin
  { Update OleObject from your controls }
  OleObject.TabWidth := LstPreview.TabWidth;
end;


procedure TPpgTabWidth.ChkUseTabsClick(Sender: TObject);
begin
  TrkTabWidth.Enabled := ChkUseTabs.Checked;
  if ChkUseTabs.Checked then
    LstPreview.TabWidth := TrkTabWidth.Position * 4
  else
    LstPreview.TabWidth := 0;
end;


procedure TPpgTabWidth.TrkTabWidthChange(Sender: TObject);
begin
  Modified;
  LstPreview.TabWidth := TrkTabWidth.Position * 4;
end;


initialization
  TActiveXPropertyPageFactory.Create(
    ComServer, TPpgTabWidth, Class_PpgTabWidth );
end.
```