Delphi by Design

One-Step ActiveX

by Ray Konopka
February/March 1998, Vol. 8, No. 6 -- Download Source Code:

Getting from VCL to ActiveX can be as little as one st you're careful and understand the limitations of the A component architecture.

Over the past three installments of "Delphi by Design," the focus has been c in Delphi 3. Along the way, we have seen how interfaces play an important particular, we have seen how they are used to manipulate objects, how t abilities of automation servers, and how Windows itself relies on them. All of serves as a foundation for the topic I'll be covering in this article and the n native Delphi components into ActiveX controls.

Not Just for Delphi Anymore

Although both Delphi 2 and 3 can use ActiveX controls, the princi architecture in Delphi has always been the Visual Component Library (VCL). I is so integrated into Delphi that they are virtually indistinguishable. As a resu more power and flexibility when native Delphi components are used.

A testament to the power of the VCL is Borland's new C++ Builder, which als as its underlying architecture. In fact, C++ Builder uses native Delphi compo without any code changes. You simply recompile the Delphi component u Builder to install it on the palette.

Because C++ Builder is based on the same VCL as Delphi, it is possible to cre Builder components using C++ instead of Object Pascal—the process is creating a native Delphi component. However, I do not recommend this appr C++ Builder can use Delphi components, Delphi cannot use C++ Builde Therefore, create your custom components in Delphi if you want to use environments. Besides, it saves me from writing a C++ version of my *Dev Delphi 3 Components* book.

In Delphi 3, we now have the ability to convert our native Delphi compor ActiveX controls that can be used in a wide variety of products, includi Builder, Internet Explorer, Visual Basic, Visual C++, and IntraBuilder.

An impressive list of products, indeed. But who wants to sacrifice t productivity of building native Delphi components and learn the inner workir ActiveX? Fortunately, with Delphi 3, you don't have to.

The Delphi ActiveX Framework

Delphi 3 introduces the Delphi ActiveX Framework (DAX), which enables ActiveX controls from native Delphi components. With the additional help o the conversion process requires little knowledge of COM and ActiveX. information that I have covered in the previous three articles will be extre understanding the DAX framework.

Creating an ActiveX control in Delphi involves creating a COM wrapper a Delphi component. The wrapper is defined as a descendant of TActiv manages a reference to a Delphi component. The Delphi component functionality of the ActiveX control. It must be a descendant of TWinCon window handle is needed to support the communication between the COM w component.

The wrapper class exposes the functionality of the Delphi component to the and servers through properties and methods. It is also responsible for manipulating, and destroying the embedded component. The COM wrapper u to events generated by the component by forwarding them to the Active (the container).

One-Step ActiveX

Still sounds pretty complicated, doesn't it? Fortunately, Delphi supplies Control Wizard that greatly simplifies the process. Let's use the wizard familiar TListBox component into an ActiveX control. First, select File|New Object Repository, then switch to the ActiveX page, as shown

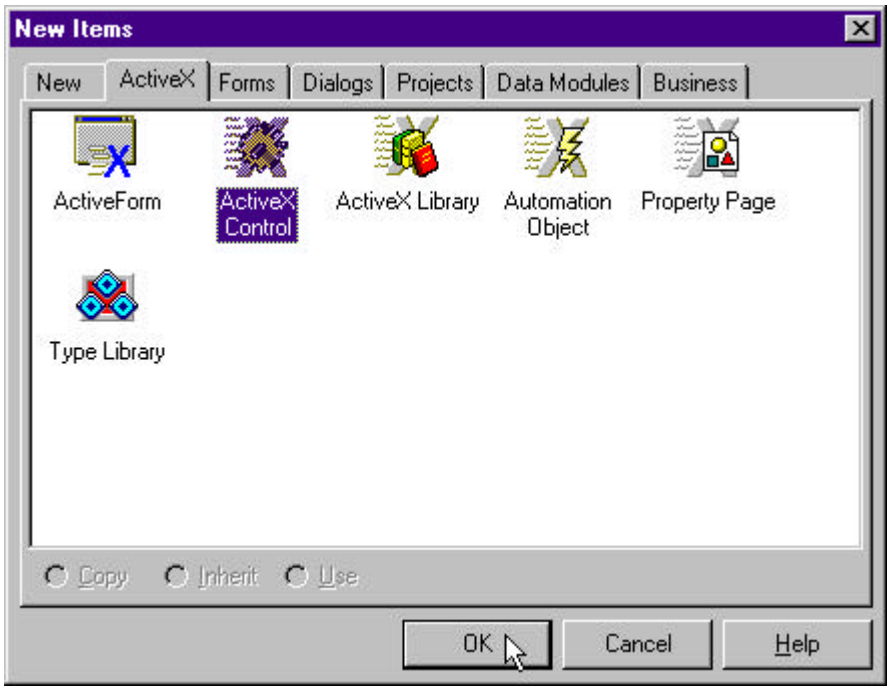

Figure 1: Starting the ActiveX Control Wizard.

Select the ActiveX Control item and click the OK button to start the A Wizard, which is shown in Figure 2. The first step is to select the componen define the functionality of the ActiveX control. The VCL Class Name combo the components currently registered with Delphi that are descendants of T that have not been removed from the list through a call to RegisterNonAct TListBox is on the list, TLabel is not because it descends from TG

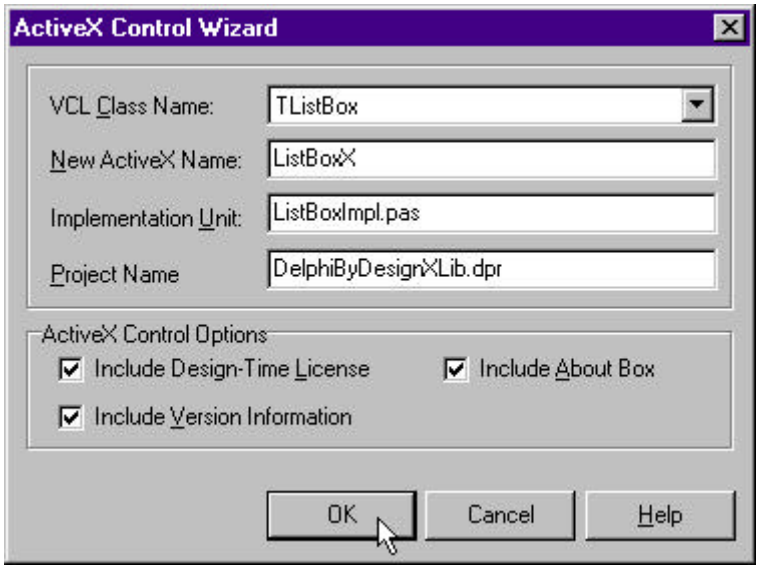

Figure 2: The ActiveX Control Wizard.

When you select a component class, the wizard automatically populates the with suggested values. The second field is used to specify a name for ActiveX control. The wizard generates a suggested name by dropping the in component type and adding an X suffix.

The third field is used to specify the name of the implementation unit, wh the TActiveXControl class descendant that defines the new ActiveX cont wizard suggests a name based on the component selected.

The fourth field is used to specify the Ac-tiveX library in which the Activ reside. When an ActiveX library that contains ActiveX controls is compiled created with the same name as the project. If there is no project open, or if library project is opened when the wizard is started, the Project Name field must be filled in with a project name. The wizard suggests a name, once a the component name. However, if you plan on adding multiple ActiveX c library, you will probably want to change the default name to something mo as I did in Figure 2.

If an ActiveX library project is opened when the wizard is started, the fourth and the new ActiveX control will be added to the currently opened project.

ActiveX Control Options

Figure 2 also shows that all of the ActiveX Control Options have been sele option specifies that a design-time license should be created for the prevents the control from being used in a design environment unless the license key for the control. With this option selected, the wizard generates control and stores it in an LIC file under the same name as the project. The a copy of this file in order to use the control in a design

The second option specifies that version information should be added to th is, the resulting OCX file. Adding version information to your library allows information about your library, such as copyright information and versio specify the details, select Project|Options and switch to the Version-Info wizard generates the new project. *(NOTE: Although version information must select this option if you plan on using the new ActiveX control in Visua required for the control to be registered correctly.)*

The final option on the ActiveX Control Wizard specifies that an about created for the new control. The about box is a separate form in its own u

the name of the control, copyright information, and an OK button. The abo displayed in design environments via the About property.

Generating the Files

Click on the OK button. If necessary, the wizard first creates an ActiveX Next, it creates an implementation unit, which is where the COM wrapper arc Delphi component is defined. The wrapper is simply a descendant of the T class.

The wizard then creates a type library and a type library interface unit. The binary file that defines the data types, interfaces, methods, and object cl ActiveX library will expose. The type library interface unit contains declarations corresponding to the information stored in the type library.

If necessary, the wizard will also create a license file, an about box form unit, and then add them to the project.

Building and Registering the Server

At this point, we can build the ActiveX library project to generate an OCX fil the ActiveX control. We can then register the library by selecting Run|R Server. However, before doing that, I recommend saving the project and source files. Although Delphi allows you to register the library without sav files to disk, this should be avoided. When an ActiveX library is registered, t its location is stored in the system registry. If you do not save the pr directory stored in the registry will be the current directory.

Problems arise if you save the ActiveX library project in a directory other specified in the registry. Specifically, Windows will not be able to load your A because it won't be able to find the library. Or worse, Windows will use an o the library residing in the old directory. If you accidentally register the Activ can remove the library from the registry by selecting Run|Unregister A

Once registered, the ListBoxX ActiveX control can be used wherever an A can be used. For example, Figure 3 shows the control being used inside V

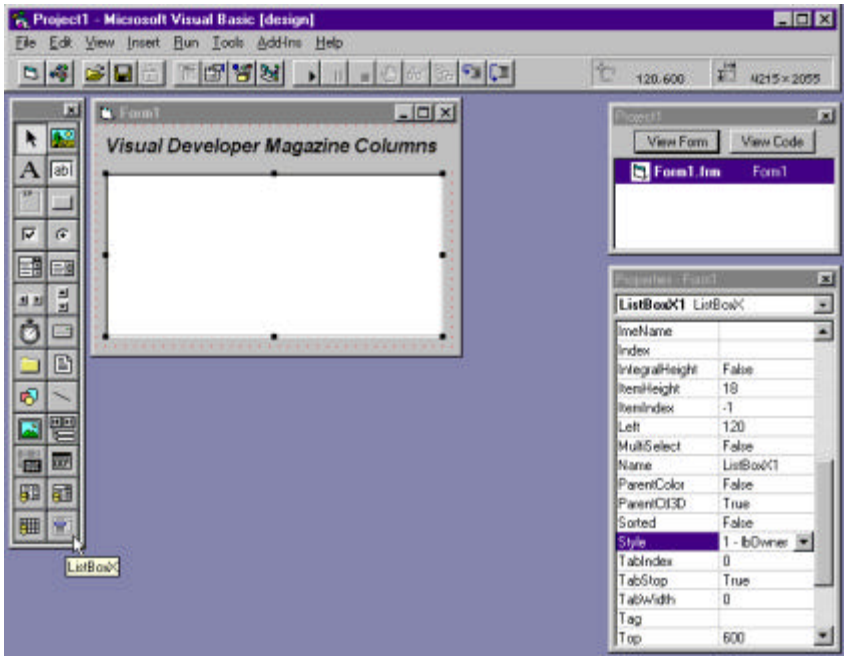

Figure 3: Using a Delphi component in VB.

(NOTE: If you change the name of the about form file from About1 to s descriptive, the next time you compile, you will receive a syntax error bec referenced in the implementation unit. Simply update the uses clause a

ActiveX Conversion Issues

Although the ListBoxX control allows us to use the TListBox component in Vi implementation of ListBoxX is not identical to that of TListBox . For OnDrawItem event defined in TListBox is not defined in ListBoxX. Re OnDrawItem event is how the TListBox component supports owner-drawn this event, the ActiveX version does not support this feature, even tho property is shown in the property inspector.

To understand why the event doesn't appear in the ActiveX version, we r the event type. In particular, we need to look at the parameters passec handlers. The OnDrawItem event is declared using the TDrawItemEvent defined as follows:

```
TDrawItemEvent = procedure( Control: TWinControl;
                    Index: Integer; Rect: TRect;
                    State: TOwnerDrawState)of object;
```

Therefore, OnDrawItem event handlers receive four parameters. The t prevents this event from appearing in the ActiveX control because OLE Au not know how to handle marshalling TRect data.

Let me explain. ActiveX controls are actually tiny OLE Automation se containers that hold ActiveX controls use Automation to communicate wit Therefore, in order for a property, method, or event to appear in the Acti the component, all parameters and return types must be *automation-comp* lists the automation-compatible types.

TABLE 1

Automation-compatible types.

Predefined Types
SmallInt, Integer, Single, Double, Currency, TDateTime, WideString, IDispatch, WordBool, Variant, OleVariant, SCode, Byte, and IUnknown
Enumerations defined in a type library
Interface types defined in a type library
Dispatch interfaces defined in a type library

Although not listed, the ActiveX Control Wizard is able to provide access to TPicture, and TStrings properties in an ActiveX control. Color properties are TColor is simply an integer value. However, for the other property types, De custom interface to handle accessing the corresponding Delphi property. Fo IStrings interface provides ActiveX access to TStrings properties.

However, just because a property is automation-compatible does not me appear in the ActiveX control. The wizard will not surface properties, meth that do not make sense for an ActiveX control. For example, the following pr converted: Height, HelpContext, Hint, Left, Name, ParentFont, P

PopupMenu, ShowHint, TabOrder, Tag, Top, and Width.

In addition, there are two other specific types of properties that the wizard an ActiveX control: *component references and data-aware properties* references are implemented as pointers and are thus not automation-comp the ActiveX model does not provide a standard way for a control to be controls in the container.

DataSource and DataField properties are not mapped to an ActiveX control components implement data-awareness differently from ActiveX controls. W to create a data-aware ActiveX control? This is certainly possible, but it requ work, which I'll cover in my next column. However, I must point out data-awareness is not the same as Delphi data-awareness. This is why non native data-aware controls appear in the ActiveX Control Wizard c

Correcting the Conversion

At this point, we have a functional ActiveX control. As you can see from Fig other list box features are available. We can even change the IsOwnerDrawFixed, but we cannot alter the items' appearance without th event—or can we?

Indeed we can, but we need to use a different approach. That is, we nee automation-compatible interface to the owner-draw capabilities of TListBox we can create a new event in the ActiveX control called OnColorItem that user the ability to change the color of individual items.

Adding a New Event

Adding a new event to an ActiveX control requires modifying the type inform the type library. There are two ways to accomplish this: by selecting th Interface menu item or by using the built-in type library editor. The Edit|Ad menu item is only enabled when the implementation unit for the ActiveX con Figure 4 shows the effect of selecting this menu item while the ListBoxImpl

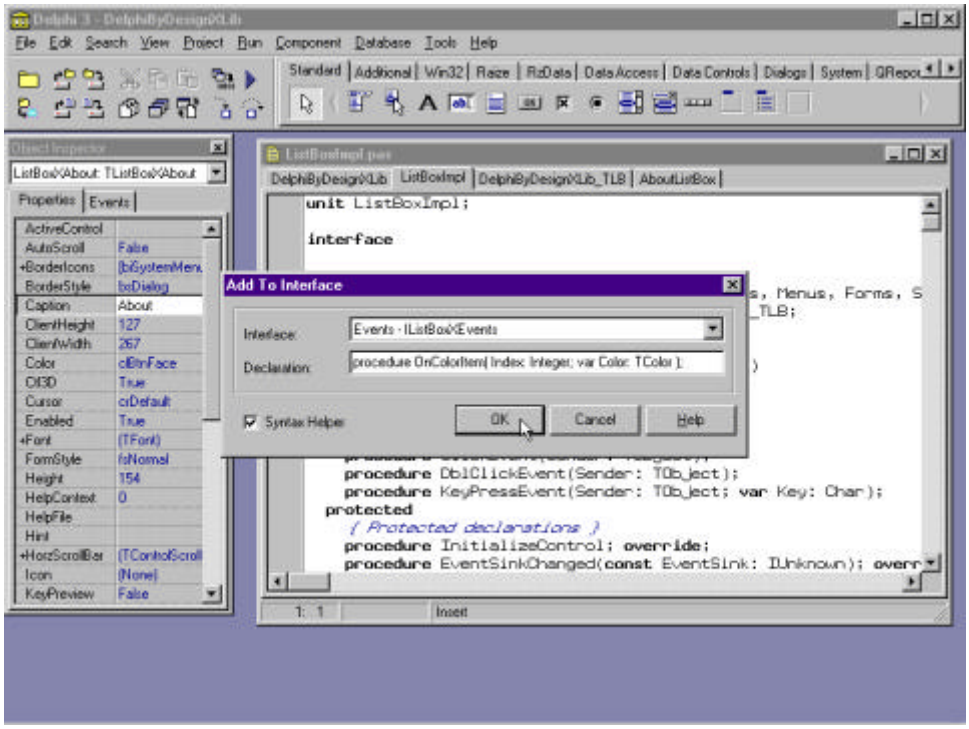

Figure 4: Adding a custom event.

From here, you must select the interface you wish to modify. Sele
Properties/Methods interface or the Events interface, depending on the ite
add. The distinction between properties/methods and events is necessary
group is controlled by a different interface. That is, the Properties/Metho
defined as a dual interface, while the Events interface is defined as a disp
you know why I went through the trouble of explaining these concepts in t
issue.

Figure 4 shows how the OnColorItem event is added to the IListBoxXEv
Defining a new event in an ActiveX interface is a bit different than defining
Delphi component. Instead of specifying a property with an event ty
TNotifyEvent, you write a procedure heading with the parameters that will
event handler. It is similar to writing an event dispatch method in a co
OnColorItem event is defined as follows:

```
procedure OnColorItem(Index: Integer; var Color: TColor );
```

After the OK button is clicked, Delphi adds the declaration to the type librar
users of the control to write event handlers for the OnColorItem event. Hov
only declared the event; we have not implemented the code to generate the
this is very similar to defining new events in a Delphi component. Decla
property and event dispatch method is only part of the process. We mus
event dispatch method at the appropriate time to generate

So, the question now is where do we generate the OnColorItem event? The
realizing that the ActiveX wrapper has access to the embedded Delp
Therefore, the TListBoxX wrapper class can create custom event handlers
events generated by the embedded Delphi component. The best place to
OnColorItem event is within the component's OnDrawItem event. **Listing 1**
listing of the ListBoxImpl unit to illustrate how this is accomplished.

The TListBoxX wrapper class defines an event handler called DrawItemEvent
the OnDrawItem event of the embedded list box. The DrawItemEvent handle
the OnDrawItem event in the InitializeControl method.

The DrawItemEvent handler is responsible for generating the OnColorIter
ActiveX control. To generate an event, the FEvents interface reference is
the OnColorItem event procedure. The Index parameter comes from the
parameter list, and the Color parameter is declared locally in the DrawItemE
OnColorItem event returns, the ItemColor variable will contain either the de
user-defined color. The ItemColor variable is then used in draw

Figure 5 shows the VB app from Figure 3 running. The code editor in Figur
event handler used to highlight an item in the list. Of course, this is no
having access to the Canvas property of the list box, but it does demonstra
that flexibility can be surfaced in the ActiveX control.

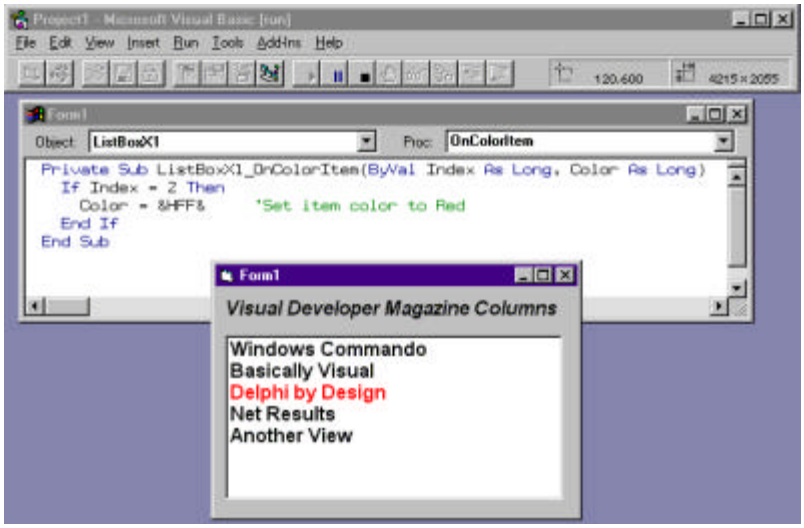

Figure 5: Using a custom event.

Attaining True One-Step ActiveX

As you can see from the previous example, it is not always easy to conv components into ActiveX controls. This is because we converted an component—in particular, a component that used some advanced features Component Library. In this respect, the ActiveX Control Framework is not as VCL. However, you must weigh this against the benefit of being able to use A in products other than Delphi and C++ Builder.

If your goal is to create an ActiveX control, there are some guidelines tl follow. First, make sure your component descends from TWinControl descendants. Graphic controls cannot be embedded within a TActiveXCont you need to provide custom painting in your component, descend from T instead.

Second, use only automation-compatible types for properties and method p will enable the ActiveX Control Wizard to convert your component's proper also force you to redesign the interface to your components.

Third, the Delphi component should be considered the source code for the A Therefore, it is better to modify the Delphi component and reconvert it ir control rather than modifying the ActiveX control's implementation unit. Of unavoidable if you don't have the source for the Delphi component.

In summary, one-step ActiveX is indeed attainable if you follow these guidelir

On the Drawing Board

Next time, we will continue discussing the process of converting Delphi co ActiveX controls. In particular, we will cover advanced features such a browsing, property pages, streaming, and deployment. 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;
    FEvents: IListBoxXEvents;
    procedure ClickEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);

    // Add a custom event handler for the OnDrawItem event
    procedure DrawItemEvent( Control: TWinControl; Index: Integer;
                        Rect: TRect; State: TOwnerDrawState );
  protected
    { Protected declarations }
    procedure InitializeControl; override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    procedure DefinePropertyPages(
      DefinePropertyPage: TDefinePropertyPage); override;
    function Get_BorderStyle: TxBorderStyle; safecall;
    function Get_Color: TColor; safecall;
    function Get_Columns: Integer; safecall;
    function Get_Ctl3D: WordBool; safecall;
    function Get_Cursor: Smallint; safecall;
    function Get_DragCursor: Smallint; safecall;
    function Get_DragMode: TxDragMode; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_ExtendedSelect: WordBool; safecall;
    function Get_Font: Font; safecall;
    function Get_ImeMode: TxImeMode; safecall;
    function Get_ImeName: WideString; safecall;
    function Get_IntegralHeight: WordBool; safecall;
    function Get_ItemHeight: Integer; safecall;
    function Get_ItemIndex: Integer; safecall;
    function Get_Items: IStrings; safecall;
    function Get_MultiSelect: WordBool; safecall;
    function Get_ParentColor: WordBool; safecall;
    function Get_ParentCtl3D: WordBool; safecall;
    function Get_SelCount: Integer; safecall;
    function Get_Sorted: WordBool; safecall;
    function Get_Style: TxListBoxStyle; safecall;
    function Get_TabWidth: Integer; safecall;
    function Get_TopIndex: Integer; safecall;
    function Get_Visible: WordBool; safecall;
    procedure AboutBox; safecall;
    procedure Clear; safecall;
    procedure Set_BorderStyle(Value: TxBorderStyle); safecall;
    procedure Set_Color(Value: TColor); safecall;
    procedure Set_Columns(Value: Integer); safecall;
    procedure Set_Ctl3D(Value: WordBool); safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    procedure Set_DragMode(Value: TxDragMode); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_ExtendedSelect(Value: WordBool); safecall;
    procedure Set_Font(const Value: Font); safecall;
```

```
    procedure Set_ImeMode(Value: TxImeMode); safecall;
    procedure Set_ImeName(const Value: WideString); safecall;
    procedure Set_IntegralHeight(Value: WordBool); safecall;
    procedure Set_ItemHeight(Value: Integer); safecall;
    procedure Set_ItemIndex(Value: Integer); safecall;
    procedure Set_Items(const Value: IStrings); safecall;
    procedure Set_MultiSelect(Value: WordBool); safecall;
    procedure Set_ParentColor(Value: WordBool); safecall;
    procedure Set_ParentCtl3D(Value: WordBool); safecall;
    procedure Set_Sorted(Value: WordBool); safecall;
    procedure Set_Style(Value: TxListBoxStyle); safecall;
    procedure Set_TabWidth(Value: Integer); safecall;
    procedure Set_TopIndex(Value: Integer); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
  end;

implementation

uses AboutListBox;

{ TListBoxX }

procedure TListBoxX.InitializeControl;
begin
  FDelphiControl := Control as TListBox;
  FDelphiControl.OnClick := ClickEvent;
  FDelphiControl.OnDblClick := DblClickEvent;
  FDelphiControl.OnKeyPress := KeyPressEvent;

  // Add a custom event handler for the OnDrawItem event
  FDelphiControl.OnDrawItem := DrawItemEvent;
end;

procedure TListBoxX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IListBoxXEvents;
end;

function TListBoxX.Get_Enabled: WordBool;
begin
  Result := FDelphiControl.Enabled;
end;

// Other Get_ Methods


procedure TListBoxX.Set_Enabled(Value: WordBool);
begin
  FDelphiControl.Enabled := Value;
end;

// Other Set_ Methods


procedure TListBoxX.DrawItemEvent( Control: TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState );
var
  ItemColor: TColor;
begin
  ItemColor := FDelphiControl.Font.Color;

  // Generate the OnColorItem ActiveX event
```

```
  if FEvents <> nil then
    FEvents.OnColorItem( Index, ItemColor );

  // Draw the item using the ItemColor
  with FDelphiControl do
  begin
    if not ( odSelected in State ) then
      Canvas.Font.Color := ItemColor;

    Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top, Items[Index]);
    Canvas.Font.Color := FDelphiControl.Font.Color;
  end;
end;


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