Creating ActiveX Controls

CHAPTER

25

IN THIS CHAPTER

For many developers, the ability to easily create ActiveX controls is one of the most compelling features Delphi brings to the table. ActiveX is a standard for programming language-independent controls that can function in a variety of environments, including Delphi, C++Builder, Visual Basic, and Internet Explorer. These controls can be as simple as a static text control or as complex as a fully functional spreadsheet or word processor. Traditionally, ActiveX controls are quite complicated and difficult to write, but Delphi brings ActiveX control creation to the masses by allowing you to convert a relatively easy-to-create VCL component or form into an ActiveX control.

This chapter will not teach you everything there is to know about ActiveX controls—that would take a thick book in its own right. What this chapter will demonstrate is how ActiveX control creation works in Delphi and how to use the Delphi wizards and framework to make Delphi-created ActiveX controls work for you.

NOTE

The capability to create ActiveX controls is provided only with the Professional and Enterprise editions of Delphi.

Why Create ActiveX Controls?

As a Delphi developer, you may be completely happy with the capabilities of native VCL components and forms, and you might be wondering why you should even bother creating ActiveX controls. There are several reasons. First, if you are a professional component developer, the payoff can be huge; by converting your VCL controls into ActiveX controls, your potential market is not merely fellow Delphi and C++Builder developers but also users of practically any Win32 development tool. Second, even if you are not a component vendor, you can take advantage of ActiveX controls to add content and functionality to World Wide Web pages.

Creating an ActiveX Control

Delphi's one-step wizards make creating an ActiveX control a simple process. However, as you will soon learn, the wizard is just the beginning if you want your controls to really shine.

To help you become familiar with Delphi's ActiveX capabilities, Figure 25.1 shows the ActiveX page of the New Items dialog, which appears when you select File, New from the main menu. Many of the items shown here will be described as this chapter progresses.

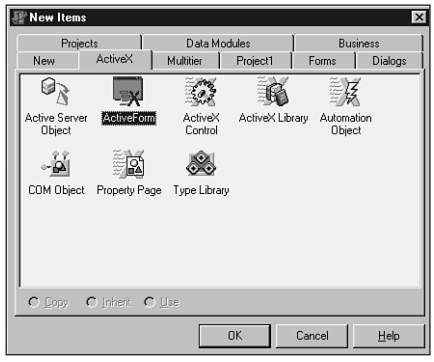

FIGURE 25.1
The ActiveX page of the New Items dialog.

The first icon in this dialog represents an ActiveForm (described later in this chapter), and you can click it to invoke a wizard that aids you in creating an ActiveForm. Note that ActiveForms are only slightly different from regular ActiveX controls, so we will refer to both generically as *ActiveX controls* throughout this chapter.

Next, you see the icon representing an ActiveX control. Clicking here will invoke the ActiveX Control Wizard, which we will describe in the next section.

The third icon represents an ActiveX library. Click this icon to create a new library project that exports the four ActiveX server functions described in Chapter 23, "COM-Based Technologies." This can be used as a starting point before adding an ActiveX control to the project.

The Automation Object Wizard, represented by the next icon, is described in Chapter 23.

The next icon is the COM Object Wizard. The wizard invoked by clicking this icon enables you to create a plain COM object. You learned about this wizard in the previous chapter when you created shell extensions.

Clicking the icon at the far right enables you to add a property page to the current project. *Property pages* allow visual editing of ActiveX controls, and you will see an example of creating a property page and integrating it into your ActiveX control project later in this chapter.

The final icon represents a type library; you can click it when you wish to add a type library to your project. Because the wizards for ActiveX controls and ActiveForms (as well as Automation objects) automatically add a type library to the project, you will rarely use this option.

The ActiveX Control Wizard

Clicking the ActiveX Control icon on the ActiveX page of the New Items dialog will invoke the ActiveX Control Wizard, which is shown in Figure 25.2.

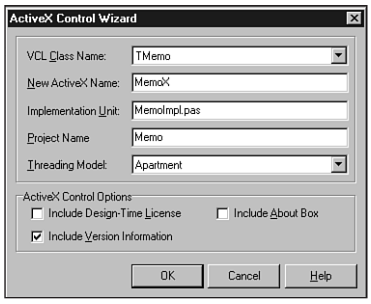

FIGURE 25.2
The ActiveX Control Wizard.

This wizard allows you to choose a VCL control class to encapsulate as an ActiveX control. Additionally, it allows you to specify the name of the ActiveX control class, the name of the file that will contain the implementation of the new ActiveX control, and the name of the project in which the ActiveX control will reside.

VCL Controls in the ActiveX Control Wizard

If you examine the list of VCL controls in the drop-down combo box in the ActiveX Control Wizard, you will notice that not all VCL components are found in the list. A VCL control must meet three criteria in order to be listed in the wizard:

- The VCL control must reside in a currently installed design package (that is, it must be on the Component Palette).
- The VCL control must descend from `TWinControl`. Currently, nonwindowed controls cannot be encapsulated as ActiveX controls.
- The VCL control must not have been excluded from the list with the `RegisterNonActiveX()` procedure. `RegisterNonActiveX()` is described in detail in the Delphi online help.

Many standard VCL components are excluded from the list because they either do not make sense as ActiveX controls or would require significant work beyond the wizard's scope in order to function as ActiveX controls. `TDBGrid` is a good example of a VCL control that does not make sense as an ActiveX control; it requires another VCL class (`TDataSource`) as a property in order to function, and this is not possible using ActiveX. `TTreeView` is an example of a control that would require significant work beyond the wizard to encapsulate as an ActiveX control, because the `TTreeView` nodes would be difficult to represent in ActiveX.

NOTE

Although the ActiveX wizard does not allow you to automatically generate an ActiveX control from a non-`TWinControl` control, it is possible to write such a control by hand using the Delphi ActiveX (DAX) framework.

ActiveX Control Options

The lower portion of the ActiveX Control Wizard dialog allows you to set certain options that will become a part of the ActiveX control. These options consist of three check boxes:

- *Make Control Licensed.* When this option is selected, a license (LIC) file will be generated along with the control project. In order for other developers to use the generated ActiveX control in a development environment, they will need to have the LIC file in addition to the ActiveX control (OCX) file.
- *Include Version Information.* When selected, this option will cause a `VersionInfo` resource to be linked into the OCX file. In addition, the string file information in the `VersionInfo` resource includes a value called `OleSelfRegister`, which is set to `1`. This setting is required for some older ActiveX control hosts, such as Visual Basic 4.0. You can edit a project's `VersionInfo` data in the VersionInfo page of the Project Options dialog.
- *Include About Box.* Select this option in order to include an "About box" dialog with your ActiveX control. The About box is usually available in ActiveX container applications by selecting an option from a local right-click menu on the ActiveX control. The About box generated is a regular Delphi form that you can edit to your liking.

How VCL Controls Are Encapsulated

After you finish describing your control in the ActiveX Control Wizard and click the OK button, the wizard goes about the task of writing the wrapper to encapsulate the selected VCL control as an ActiveX control. The end result is an ActiveX library project that includes a working ActiveX control, but a lot of interesting stuff is going on behind the scenes. Here is a description of the steps involved in encapsulating a VCL control as an ActiveX control:

1. The wizard determines which unit contains the VCL control. That unit is then handed to the compiler, and the compiler generates special symbolic information for the VCL control's properties, methods, and events.
2. A type library is created for the project. It contains an interface to hold properties and methods, a dispinterface to hold events, and a coclass to represent the ActiveX control.

3. The wizard iterates over all the symbol information for the VCL control, adding qualified properties and methods to the interface in the type library and qualified events to the dispinterface.

NOTE

The description of step 3 begs the following question: What constitutes a *qualified* property, method, or event for inclusion in the type library? In order to qualify for inclusion in the type library, properties must be of an Automation-compatible type, and the parameters and return values of the methods and events must also be of an Automation-compatible type. Recall from Chapter 23, "COM-Based Technologies," that Automation-compatible types include `Byte`, `SmallInt`, `Integer`, `Single`, `Double`, `Currency`, `TDateTime`, `WideString`, `WordBool`, `PSafeArray`, `TDecimal`, `OleVariant`, `IUnknown`, `IDispatch`.

However, there are exceptions to this rule. In addition to Automation-compatible types, parameters of type `TStrings`, `TPicture`, and `TFont` are also permitted. For these types, the wizard will employ special adapter objects that allow them to be wrapped with an ActiveX-compatible `IDispatch` or dispinterface.

4. Once all the qualifying properties, methods, and events have been added, the type library editor generates a file that is an Object Pascal translation of the type library contents.
5. The wizard then generates the implementation file for the ActiveX control. This implementation file contains a `TActiveXControl` object that implements the interface described in the type library. The wizard automatically writes *forwarders* for interface properties and methods. These forwarder methods forward method calls from the ActiveX control wrapper into the control, and they forward events from the VCL control out to the ActiveX control.

To help illustrate what we are describing here, we have provided the following listings. They belong to an ActiveX control project created from a `TMemo` VCL control. This project is saved as `Memo.dpr`. Listing 25.1 shows the project file, Listing 25.2 shows the type library file, and Listing 25.3 shows the implementation file generated for the control. Also, Figure 25.3 shows the contents of the type library editor.

LISTING 25.1 The Project File: `Memo.dpr`

```
library Memo;

uses
  ComServ,
  Memo_TLB in 'Memo_TLB.pas',
```

```
  MemoImpl in 'MemoImpl.pas' {MemoX: CoClass},
  About in 'About.pas' {MemoXAbout};

{$E ocx}

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.TLB}

{$R *.RES}

begin
end.
```

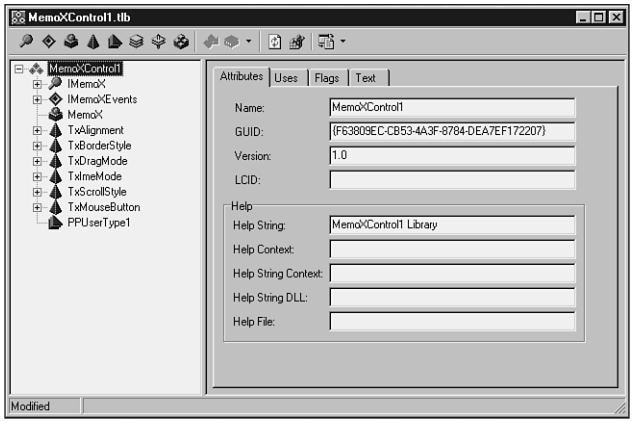

FIGURE 25.3
Memo, *as shown in the type library editor.*

LISTING 25.2 The Type Library File: `Memo_TLB.pas`

```
unit Memo_TLB;

// ************************************************************************ //
// WARNING
// -------
// The types declared in this file were generated from data read from a
```

continues

LISTING 25.2 Continued

```
// Type Library. If this type library is explicitly or indirectly (via
// another type library referring to this type library) re-imported, or the
// 'Refresh' command of the Type Library Editor activated while editing the
// Type Library, the contents of this file will be regenerated and all
// manual modifications will be lost.
// ************************************************************************ //

// PASTLWTR : $Revision:   1.88  $
// File generated on 8/23/99 12:22:29 AM from Type Library described below.

// ************************************************************************ //
// NOTE:
// Items guarded by $IFDEF_LIVE_SERVER_AT_DESIGN_TIME are used by properties
// which return objects that may need to be explicitly created via a function
// call prior to any access via the property. These items have been disabled
// in order to prevent accidental use from within the object inspector. You
// may enable them by defining LIVE_SERVER_AT_DESIGN_TIME or by selectively
// removing them from the $IFDEF blocks. However, such items must still be
// programmatically created via a method of the appropriate CoClass before
// they can be used.
// ************************************************************************ //
// Type Lib: X:\work\d5dg\code\Ch25\Memo\Memo.tlb (1)
// IID\LCID: {0DB4686F-09C5-11D2-AE5C-00A024E3867F}\0
// Helpfile:
// DepndLst:
//   (1) v2.0 stdole, (C:\WINDOWS\SYSTEM\STDOLE2.TLB)
//   (2) v4.0 StdVCL, (C:\WINDOWS\SYSTEM\STDVCL40.DLL)
// ************************************************************************ //
{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
interface

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

// ************************************************************************//
// GUIDS declared in the TypeLibrary. Following prefixes are used:
//   Type Libraries     : LIBID_xxxx
//   CoClasses          : CLASS_xxxx
//   DISPInterfaces     : DIID_xxxx
//   Non-DISP interfaces: IID_xxxx
// ************************************************************************//
const
  // TypeLibrary Major and minor versions
  MemoMajorVersion = 1;
  MemoMinorVersion = 0;
```

```
  LIBID_Memo: TGUID = '{0DB4686F-09C5-11D2-AE5C-00A024E3867F}';

  IID_IMemoX: TGUID = '{0DB46870-09C5-11D2-AE5C-00A024E3867F}';
  DIID_IMemoXEvents: TGUID = '{0DB46872-09C5-11D2-AE5C-00A024E3867F}';
  CLASS_MemoX: TGUID = '{0DB46874-09C5-11D2-AE5C-00A024E3867F}';

// *********************************************************************//
// Declaration of Enumerations defined in Type Library
// *********************************************************************//
// Constants for enum TxAlignment
type
  TxAlignment = TOleEnum;
const
  taLeftJustify = $00000000;
  taRightJustify = $00000001;
  taCenter = $00000002;

// Constants for enum TxBiDiMode
type
  TxBiDiMode = TOleEnum;
const
  bdLeftToRight = $00000000;
  bdRightToLeft = $00000001;
  bdRightToLeftNoAlign = $00000002;
  bdRightToLeftReadingOnly = $00000003;

// Constants for enum TxBorderStyle
type
  TxBorderStyle = TOleEnum;
const
  bsNone = $00000000;
  bsSingle = $00000001;

// Constants for enum TxDragMode
type
  TxDragMode = TOleEnum;
const
  dmManual = $00000000;
  dmAutomatic = $00000001;

// Constants for enum TxImeMode
type
  TxImeMode = TOleEnum;
const
  imDisable = $00000000;
  imClose = $00000001;
```

continues

LISTING 25.2 Continued

```
  imOpen = $00000002;
  imDontCare = $00000003;
  imSAlpha = $00000004;
  imAlpha = $00000005;
  imHira = $00000006;
  imSKata = $00000007;
  imKata = $00000008;
  imChinese = $00000009;
  imSHanguel = $0000000A;
  imHanguel = $0000000B;

// Constants for enum TxScrollStyle
type
  TxScrollStyle = TOleEnum;
const
  ssNone = $00000000;
  ssHorizontal = $00000001;
  ssVertical = $00000002;
  ssBoth = $00000003;

// Constants for enum TxMouseButton
type
  TxMouseButton = TOleEnum;
const
  mbLeft = $00000000;
  mbRight = $00000001;
  mbMiddle = $00000002;

type

// *********************************************************************//
// Forward declaration of types defined in TypeLibrary
// *********************************************************************//
  IMemoX = interface;
  IMemoXDisp = dispinterface;
  IMemoXEvents = dispinterface;

// *********************************************************************//
// Declaration of CoClasses defined in Type Library
// (NOTE: Here we map each CoClass to its Default Interface)
// *********************************************************************//
  MemoX = IMemoX;


// *********************************************************************//
// Interface: IMemoX
```

```
// Flags:     (4416) Dual OleAutomation Dispatchable
// GUID:      {0DB46870-09C5-11D2-AE5C-00A024E3867F}
// ************************************************************************//
  IMemoX = interface(IDispatch)
    ['{0DB46870-09C5-11D2-AE5C-00A024E3867F}']
    function  Get_Alignment: TxAlignment; safecall;
    procedure Set_Alignment(Value: TxAlignment); safecall;
    function  Get_BiDiMode: TxBiDiMode; safecall;
    procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
    function  Get_BorderStyle: TxBorderStyle; safecall;
    procedure Set_BorderStyle(Value: TxBorderStyle); safecall;
    function  Get_Color: OLE_COLOR; safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    function  Get_Ctl3D: WordBool; safecall;
    procedure Set_Ctl3D(Value: WordBool); safecall;
    function  Get_DragCursor: Smallint; safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    function  Get_DragMode: TxDragMode; safecall;
    procedure Set_DragMode(Value: TxDragMode); safecall;
    function  Get_Enabled: WordBool; safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    function  Get_Font: IFontDisp; safecall;
    procedure _Set_Font(const Value: IFontDisp); safecall;
    procedure Set_Font(var Value: IFontDisp); safecall;
    function  Get_HideSelection: WordBool; safecall;
    procedure Set_HideSelection(Value: WordBool); safecall;
    function  Get_ImeMode: TxImeMode; safecall;
    procedure Set_ImeMode(Value: TxImeMode); safecall;
    function  Get_ImeName: WideString; safecall;
    procedure Set_ImeName(const Value: WideString); safecall;
    function  Get_MaxLength: Integer; safecall;
    procedure Set_MaxLength(Value: Integer); safecall;
    function  Get_OEMConvert: WordBool; safecall;
    procedure Set_OEMConvert(Value: WordBool); safecall;
    function  Get_ParentColor: WordBool; safecall;
    procedure Set_ParentColor(Value: WordBool); safecall;
    function  Get_ParentCtl3D: WordBool; safecall;
    procedure Set_ParentCtl3D(Value: WordBool); safecall;
    function  Get_ParentFont: WordBool; safecall;
    procedure Set_ParentFont(Value: WordBool); safecall;
    function  Get_ReadOnly: WordBool; safecall;
    procedure Set_ReadOnly(Value: WordBool); safecall;
    function  Get_ScrollBars: TxScrollStyle; safecall;
    procedure Set_ScrollBars(Value: TxScrollStyle); safecall;
    function  Get_Visible: WordBool; safecall;
    procedure Set_Visible(Value: WordBool); safecall;
```

continues

LISTING 25.2 Continued

```
    function  Get_WantReturns: WordBool; safecall;
    procedure Set_WantReturns(Value: WordBool); safecall;
    function  Get_WantTabs: WordBool; safecall;
    procedure Set_WantTabs(Value: WordBool); safecall;
    function  Get_WordWrap: WordBool; safecall;
    procedure Set_WordWrap(Value: WordBool); safecall;
    function  GetControlsAlignment: TxAlignment; safecall;
    procedure Clear; safecall;
    procedure ClearSelection; safecall;
    procedure CopyToClipboard; safecall;
    procedure CutToClipboard; safecall;
    procedure PasteFromClipboard; safecall;
    procedure Undo; safecall;
    procedure ClearUndo; safecall;
    procedure SelectAll; safecall;
    function  Get_CanUndo: WordBool; safecall;
    function  Get_Modified: WordBool; safecall;
    procedure Set_Modified(Value: WordBool); safecall;
    function  Get_SelLength: Integer; safecall;
    procedure Set_SelLength(Value: Integer); safecall;
    function  Get_SelStart: Integer; safecall;
    procedure Set_SelStart(Value: Integer); safecall;
    function  Get_SelText: WideString; safecall;
    procedure Set_SelText(const Value: WideString); safecall;
    function  Get_Text: WideString; safecall;
    procedure Set_Text(const Value: WideString); safecall;
    function  Get_DoubleBuffered: WordBool; safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure FlipChildren(AllLevels: WordBool); safecall;
    function  DrawTextBiDiModeFlags(Flags: Integer): Integer; safecall;
    function  DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
    procedure InitiateAction; safecall;
    function  IsRightToLeft: WordBool; safecall;
    function  UseRightToLeftAlignment: WordBool; safecall;
    function  UseRightToLeftReading: WordBool; safecall;
    function  UseRightToLeftScrollBar: WordBool; safecall;
    function  Get_Cursor: Smallint; safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    function  ClassNameIs(const Name: WideString): WordBool; safecall;
    procedure AboutBox; safecall;
    property Alignment: TxAlignment read Get_Alignment write Set_Alignment;
    property BiDiMode: TxBiDiMode read Get_BiDiMode write Set_BiDiMode;
    property BorderStyle: TxBorderStyle read Get_BorderStyle write
      Set_BorderStyle;
    property Color: OLE_COLOR read Get_Color write Set_Color;
```

```
    property Ctl3D: WordBool read Get_Ctl3D write Set_Ctl3D;
    property DragCursor: Smallint read Get_DragCursor write Set_DragCursor;
    property DragMode: TxDragMode read Get_DragMode write Set_DragMode;
    property Enabled: WordBool read Get_Enabled write Set_Enabled;
    property Font: IFontDisp read Get_Font write _Set_Font;
    property HideSelection: WordBool read Get_HideSelection write
      Set_HideSelection;
    property ImeMode: TxImeMode read Get_ImeMode write Set_ImeMode;
    property ImeName: WideString read Get_ImeName write Set_ImeName;
    property MaxLength: Integer read Get_MaxLength write Set_MaxLength;
    property OEMConvert: WordBool read Get_OEMConvert write Set_OEMConvert;
    property ParentColor: WordBool read Get_ParentColor write Set_ParentColor;
    property ParentCtl3D: WordBool read Get_ParentCtl3D write Set_ParentCtl3D;
    property ParentFont: WordBool read Get_ParentFont write Set_ParentFont;
    property ReadOnly: WordBool read Get_ReadOnly write Set_ReadOnly;
    property ScrollBars: TxScrollStyle read Get_ScrollBars write
Set_ScrollBars;
    property Visible: WordBool read Get_Visible write Set_Visible;
    property WantReturns: WordBool read Get_WantReturns write Set_WantReturns;
    property WantTabs: WordBool read Get_WantTabs write Set_WantTabs;
    property WordWrap: WordBool read Get_WordWrap write Set_WordWrap;
    property CanUndo: WordBool read Get_CanUndo;
    property Modified: WordBool read Get_Modified write Set_Modified;
    property SelLength: Integer read Get_SelLength write Set_SelLength;
    property SelStart: Integer read Get_SelStart write Set_SelStart;
    property SelText: WideString read Get_SelText write Set_SelText;
    property Text: WideString read Get_Text write Set_Text;
    property DoubleBuffered: WordBool read Get_DoubleBuffered write
      Set_DoubleBuffered;
    property Cursor: Smallint read Get_Cursor write Set_Cursor;
  end;

// ******************************************************************//
// DispIntf:  IMemoXDisp
// Flags:     (4416) Dual OleAutomation Dispatchable
// GUID:      {0DB46870-09C5-11D2-AE5C-00A024E3867F}
// ******************************************************************//
  IMemoXDisp = dispinterface
    ['{0DB46870-09C5-11D2-AE5C-00A024E3867F}']
    property Alignment: TxAlignment dispid 1;
    property BiDiMode: TxBiDiMode dispid 2;
    property BorderStyle: TxBorderStyle dispid 3;
    property Color: OLE_COLOR dispid -501;
    property Ctl3D: WordBool dispid 4;
    property DragCursor: Smallint dispid 5;
    property DragMode: TxDragMode dispid 6;
```

continues

LISTING 25.2 Continued

```
    property Enabled: WordBool dispid -514;
    property Font: IFontDisp dispid -512;
    property HideSelection: WordBool dispid 7;
    property ImeMode: TxImeMode dispid 8;
    property ImeName: WideString dispid 9;
    property MaxLength: Integer dispid 10;
    property OEMConvert: WordBool dispid 11;
    property ParentColor: WordBool dispid 12;
    property ParentCtl3D: WordBool dispid 13;
    property ParentFont: WordBool dispid 14;
    property ReadOnly: WordBool dispid 15;
    property ScrollBars: TxScrollStyle dispid 16;
    property Visible: WordBool dispid 17;
    property WantReturns: WordBool dispid 18;
    property WantTabs: WordBool dispid 19;
    property WordWrap: WordBool dispid 20;
    function  GetControlsAlignment: TxAlignment; dispid 21;
    procedure Clear; dispid 22;
    procedure ClearSelection; dispid 23;
    procedure CopyToClipboard; dispid 24;
    procedure CutToClipboard; dispid 25;
    procedure PasteFromClipboard; dispid 27;
    procedure Undo; dispid 28;
    procedure ClearUndo; dispid 29;
    procedure SelectAll; dispid 31;
    property CanUndo: WordBool readonly dispid 33;
    property Modified: WordBool dispid 34;
    property SelLength: Integer dispid 35;
    property SelStart: Integer dispid 36;
    property SelText: WideString dispid 37;
    property Text: WideString dispid -517;
    property DoubleBuffered: WordBool dispid 39;
    procedure FlipChildren(AllLevels: WordBool); dispid 40;
    function  DrawTextBiDiModeFlags(Flags: Integer): Integer; dispid 43;
    function  DrawTextBiDiModeFlagsReadingOnly: Integer; dispid 44;
    procedure InitiateAction; dispid 46;
    function  IsRightToLeft: WordBool; dispid 47;
    function  UseRightToLeftAlignment: WordBool; dispid 52;
    function  UseRightToLeftReading: WordBool; dispid 53;
    function  UseRightToLeftScrollBar: WordBool; dispid 54;
    property Cursor: Smallint dispid 55;
    function  ClassNameIs(const Name: WideString): WordBool; dispid 59;
    procedure AboutBox; dispid -552;
  end;
```

```
// ************************************************************************//
// DispIntf:  IMemoXEvents
// Flags:     (4096) Dispatchable
// GUID:      {0DB46872-09C5-11D2-AE5C-00A024E3867F}
// ************************************************************************//
  IMemoXEvents = dispinterface
    ['{0DB46872-09C5-11D2-AE5C-00A024E3867F}']
    procedure OnChange; dispid 1;
    procedure OnClick; dispid 2;
    procedure OnDblClick; dispid 3;
    procedure OnKeyPress(var Key: Smallint); dispid 9;
  end;


// ************************************************************************//
// OLE Control Proxy class declaration
// Control Name     : TMemoX
// Help String      : MemoX Control
// Default Interface: IMemoX
// Def. Intf. DISP? : No
// Event   Interface: IMemoXEvents
// TypeFlags        : (34) CanCreate Control
// ************************************************************************//
  TMemoXOnKeyPress = procedure(Sender: TObject; var Key: Smallint) of object;

  TMemoX = class(TOleControl)
  private
    FOnChange: TNotifyEvent;
    FOnClick: TNotifyEvent;
    FOnDblClick: TNotifyEvent;
    FOnKeyPress: TMemoXOnKeyPress;
    FIntf: IMemoX;
    function  GetControlInterface: IMemoX;
  protected
    procedure CreateControl;
    procedure InitControlData; override;
  public
    function  GetControlsAlignment: TxAlignment;
    procedure Clear;
    procedure ClearSelection;
    procedure CopyToClipboard;
    procedure CutToClipboard;
    procedure PasteFromClipboard;
    procedure Undo;
    procedure ClearUndo;
    procedure SelectAll;
```

continues

LISTING 25.2 Continued

```
    procedure FlipChildren(AllLevels: WordBool);
    function  DrawTextBiDiModeFlags(Flags: Integer): Integer;
    function  DrawTextBiDiModeFlagsReadingOnly: Integer;
    procedure InitiateAction;
    function  IsRightToLeft: WordBool;
    function  UseRightToLeftAlignment: WordBool;
    function  UseRightToLeftReading: WordBool;
    function  UseRightToLeftScrollBar: WordBool;
    function  ClassNameIs(const Name: WideString): WordBool;
    procedure AboutBox;
    property  ControlInterface: IMemoX read GetControlInterface;
    property  DefaultInterface: IMemoX read GetControlInterface;
    property CanUndo: WordBool index 33 read GetWordBoolProp;
    property Modified: WordBool index 34 read GetWordBoolProp write
      SetWordBoolProp;
    property SelLength: Integer index 35 read GetIntegerProp write
      SetIntegerProp;
    property SelStart: Integer index 36 read GetIntegerProp write
      SetIntegerProp;
    property SelText: WideString index 37 read GetWideStringProp write
      SetWideStringProp;
    property Text: WideString index -517 read GetWideStringProp write
      SetWideStringProp;
    property DoubleBuffered: WordBool index 39 read GetWordBoolProp write
      SetWordBoolProp;
  published
    property Alignment: TOleEnum index 1 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property BiDiMode: TOleEnum index 2 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property BorderStyle: TOleEnum index 3 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property Color: TColor index -501 read GetTColorProp write
      SetTColorProp stored False;
    property Ctl3D: WordBool index 4 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property DragCursor: Smallint index 5 read GetSmallintProp write
      SetSmallintProp stored False;
    property DragMode: TOleEnum index 6 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property Enabled: WordBool index -514 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property Font: TFont index -512 read GetTFontProp write SetTFontProp
      stored False;
    property HideSelection: WordBool index 7 read GetWordBoolProp write
```

```
      SetWordBoolProp stored False;
    property ImeMode: TOleEnum index 8 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property ImeName: WideString index 9 read GetWideStringProp write
      SetWideStringProp stored False;
    property MaxLength: Integer index 10 read GetIntegerProp write
      SetIntegerProp stored False;
    property OEMConvert: WordBool index 11 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property ParentColor: WordBool index 12 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property ParentCtl3D: WordBool index 13 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property ParentFont: WordBool index 14 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property ReadOnly: WordBool index 15 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property ScrollBars: TOleEnum index 16 read GetTOleEnumProp write
      SetTOleEnumProp stored False;
    property Visible: WordBool index 17 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property WantReturns: WordBool index 18 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property WantTabs: WordBool index 19 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property WordWrap: WordBool index 20 read GetWordBoolProp write
      SetWordBoolProp stored False;
    property Cursor: Smallint index 55 read GetSmallintProp write
      SetSmallintProp stored False;
    property OnChange: TNotifyEvent read FOnChange write FOnChange;
    property OnClick: TNotifyEvent read FOnClick write FOnClick;
    property OnDblClick: TNotifyEvent read FOnDblClick write FOnDblClick;
    property OnKeyPress: TMemoXOnKeyPress read FOnKeyPress write FOnKeyPress;
  end;

procedure Register;

implementation

uses ComObj;

procedure TMemoX.InitControlData;
const
  CEventDispIDs: array [0..3] of DWORD = (
    $00000001, $00000002, $00000003, $00000009);
  CTFontIDs: array [0..0] of DWORD = (
```

continues

LISTING 25.2 Continued

```
    $FFFFFE00);
  CControlData: TControlData2 = (
    ClassID: '{0DB46874-09C5-11D2-AE5C-00A024E3867F}';
    EventIID: '{0DB46872-09C5-11D2-AE5C-00A024E3867F}';
    EventCount: 4;
    EventDispIDs: @CEventDispIDs;
    LicenseKey: nil (*HR:$80040154*);
    Flags: $0000002D;
    Version: 401;
    FontCount: 1;
    FontIDs: @CTFontIDs);
begin
  ControlData := @CControlData;
  TControlData2(CControlData).FirstEventOfs := Cardinal(@@FOnChange) -
    Cardinal(Self);
end;

procedure TMemoX.CreateControl;

  procedure DoCreate;
  begin
    FIntf := IUnknown(OleObject) as IMemoX;
  end;

begin
  if FIntf = nil then DoCreate;
end;

function TMemoX.GetControlInterface: IMemoX;
begin
  CreateControl;
  Result := FIntf;
end;

function  TMemoX.GetControlsAlignment: TxAlignment;
begin
  Result := DefaultInterface.GetControlsAlignment;
end;

procedure TMemoX.Clear;
begin
  DefaultInterface.Clear;
end;

procedure TMemoX.ClearSelection;
```

```
begin
  DefaultInterface.ClearSelection;
end;

procedure TMemoX.CopyToClipboard;
begin
  DefaultInterface.CopyToClipboard;
end;

procedure TMemoX.CutToClipboard;
begin
  DefaultInterface.CutToClipboard;
end;

procedure TMemoX.PasteFromClipboard;
begin
  DefaultInterface.PasteFromClipboard;
end;

procedure TMemoX.Undo;
begin
  DefaultInterface.Undo;
end;

procedure TMemoX.ClearUndo;
begin
  DefaultInterface.ClearUndo;
end;

procedure TMemoX.SelectAll;
begin
  DefaultInterface.SelectAll;
end;

procedure TMemoX.FlipChildren(AllLevels: WordBool);
begin
  DefaultInterface.FlipChildren(AllLevels);
end;

function  TMemoX.DrawTextBiDiModeFlags(Flags: Integer): Integer;
begin
  Result := DefaultInterface.DrawTextBiDiModeFlags(Flags);
end;

function  TMemoX.DrawTextBiDiModeFlagsReadingOnly: Integer;
begin
```

continues

LISTING 25.2 Continued

```
  Result := DefaultInterface.DrawTextBiDiModeFlagsReadingOnly;
end;

procedure TMemoX.InitiateAction;
begin
  DefaultInterface.InitiateAction;
end;

function  TMemoX.IsRightToLeft: WordBool;
begin
  Result := DefaultInterface.IsRightToLeft;
end;

function  TMemoX.UseRightToLeftAlignment: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftAlignment;
end;

function  TMemoX.UseRightToLeftReading: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftReading;
end;

function  TMemoX.UseRightToLeftScrollBar: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftScrollBar;
end;

function  TMemoX.ClassNameIs(const Name: WideString): WordBool;
begin
  Result := DefaultInterface.ClassNameIs(Name);
end;

procedure TMemoX.AboutBox;
begin
  DefaultInterface.AboutBox;
end;

procedure Register;
begin
  RegisterComponents('ActiveX',[TMemoX]);
end;

end.
```

NOTE

If you examine the code in Listing 25.2 carefully, you will notice that, in addition to type library information, `Memo_TLB.pas` also contains a class called `TMemoX`, which is the `TOleControl` wrapper for the ActiveX control. This enables you to add a Delphi-created ActiveX control to the palette simply by adding the generated *xxx*`_TLB` unit to a design package.

LISTING 25.3 The Implementation File: `MemoImpl.pas`

```
unit MemoImpl;

interface

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

type
  TMemoX = class(TActiveXControl, IMemoX)
  private
    { Private declarations }
    FDelphiControl: TMemo;
    FEvents: IMemoXEvents;
    procedure ChangeEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
  protected
    { Protected declarations }
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
      override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    procedure InitializeControl; override;
    function ClassNameIs(const Name: WideString): WordBool; safecall;
    function DrawTextBiDiModeFlags(Flags: Integer): Integer; safecall;
    function DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
    function Get_Alignment: TxAlignment; safecall;
    function Get_BiDiMode: TxBiDiMode; safecall;
    function Get_BorderStyle: TxBorderStyle; safecall;
    function Get_CanUndo: WordBool; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_Ctl3D: WordBool; safecall;
```

continues

LISTING 25.3 Continued

```
function Get_Cursor: Smallint; safecall;
function Get_DoubleBuffered: WordBool; safecall;
function Get_DragCursor: Smallint; safecall;
function Get_DragMode: TxDragMode; safecall;
function Get_Enabled: WordBool; safecall;
function Get_Font: IFontDisp; safecall;
function Get_HideSelection: WordBool; safecall;
function Get_ImeMode: TxImeMode; safecall;
function Get_ImeName: WideString; safecall;
function Get_MaxLength: Integer; safecall;
function Get_Modified: WordBool; safecall;
function Get_OEMConvert: WordBool; safecall;
function Get_ParentColor: WordBool; safecall;
function Get_ParentCtl3D: WordBool; safecall;
function Get_ParentFont: WordBool; safecall;
function Get_ReadOnly: WordBool; safecall;
function Get_ScrollBars: TxScrollStyle; safecall;
function Get_SelLength: Integer; safecall;
function Get_SelStart: Integer; safecall;
function Get_SelText: WideString; safecall;
function Get_Text: WideString; safecall;
function Get_Visible: WordBool; safecall;
function Get_WantReturns: WordBool; safecall;
function Get_WantTabs: WordBool; safecall;
function Get_WordWrap: WordBool; safecall;
function GetControlsAlignment: TxAlignment; safecall;
function IsRightToLeft: WordBool; safecall;
function UseRightToLeftAlignment: WordBool; safecall;
function UseRightToLeftReading: WordBool; safecall;
function UseRightToLeftScrollBar: WordBool; safecall;
procedure _Set_Font(const Value: IFontDisp); safecall;
procedure AboutBox; safecall;
procedure Clear; safecall;
procedure ClearSelection; safecall;
procedure ClearUndo; safecall;
procedure CopyToClipboard; safecall;
procedure CutToClipboard; safecall;
procedure FlipChildren(AllLevels: WordBool); safecall;
procedure InitiateAction; safecall;
procedure PasteFromClipboard; safecall;
procedure SelectAll; safecall;
procedure Set_Alignment(Value: TxAlignment); safecall;
procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
procedure Set_BorderStyle(Value: TxBorderStyle); safecall;
procedure Set_Color(Value: OLE_COLOR); safecall;
```

```
    procedure Set_Ctl3D(Value: WordBool); safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    procedure Set_DragMode(Value: TxDragMode); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(var Value: IFontDisp); safecall;
    procedure Set_HideSelection(Value: WordBool); safecall;
    procedure Set_ImeMode(Value: TxImeMode); safecall;
    procedure Set_ImeName(const Value: WideString); safecall;
    procedure Set_MaxLength(Value: Integer); safecall;
    procedure Set_Modified(Value: WordBool); safecall;
    procedure Set_OEMConvert(Value: WordBool); safecall;
    procedure Set_ParentColor(Value: WordBool); safecall;
    procedure Set_ParentCtl3D(Value: WordBool); safecall;
    procedure Set_ParentFont(Value: WordBool); safecall;
    procedure Set_ReadOnly(Value: WordBool); safecall;
    procedure Set_ScrollBars(Value: TxScrollStyle); safecall;
    procedure Set_SelLength(Value: Integer); safecall;
    procedure Set_SelStart(Value: Integer); safecall;
    procedure Set_SelText(const Value: WideString); safecall;
    procedure Set_Text(const Value: WideString); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    procedure Set_WantReturns(Value: WordBool); safecall;
    procedure Set_WantTabs(Value: WordBool); safecall;
    procedure Set_WordWrap(Value: WordBool); safecall;
    procedure Undo; safecall;
  end;

implementation

uses ComObj, About;

{ TMemoX }

procedure TMemoX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  { Define property pages here.  Property pages are defined by calling
    DefinePropertyPage with the class id of the page.  For example,
      DefinePropertyPage(Class_MemoXPage); }
end;

procedure TMemoX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IMemoXEvents;
end;
```

continues

LISTING 25.3 Continued

```
procedure TMemoX.InitializeControl;
begin
  FDelphiControl := Control as TMemo;
  FDelphiControl.OnChange := ChangeEvent;
  FDelphiControl.OnClick := ClickEvent;
  FDelphiControl.OnDblClick := DblClickEvent;
  FDelphiControl.OnKeyPress := KeyPressEvent;
end;

function TMemoX.ClassNameIs(const Name: WideString): WordBool;
begin
  Result := FDelphiControl.ClassNameIs(Name);
end;

function TMemoX.DrawTextBiDiModeFlags(Flags: Integer): Integer;
begin
  Result := FDelphiControl.DrawTextBiDiModeFlags(Flags);
end;

function TMemoX.DrawTextBiDiModeFlagsReadingOnly: Integer;
begin
  Result := FDelphiControl.DrawTextBiDiModeFlagsReadingOnly;
end;

function TMemoX.Get_Alignment: TxAlignment;
begin
  Result := Ord(FDelphiControl.Alignment);
end;

function TMemoX.Get_BiDiMode: TxBiDiMode;
begin
  Result := Ord(FDelphiControl.BiDiMode);
end;

function TMemoX.Get_BorderStyle: TxBorderStyle;
begin
  Result := Ord(FDelphiControl.BorderStyle);
end;

function TMemoX.Get_CanUndo: WordBool;
begin
  Result := FDelphiControl.CanUndo;
end;

function TMemoX.Get_Color: OLE_COLOR;
begin
```

```
  Result := OLE_COLOR(FDelphiControl.Color);
end;

function TMemoX.Get_Ctl3D: WordBool;
begin
  Result := FDelphiControl.Ctl3D;
end;

function TMemoX.Get_Cursor: Smallint;
begin
  Result := Smallint(FDelphiControl.Cursor);
end;

function TMemoX.Get_DoubleBuffered: WordBool;
begin
  Result := FDelphiControl.DoubleBuffered;
end;

function TMemoX.Get_DragCursor: Smallint;
begin
  Result := Smallint(FDelphiControl.DragCursor);
end;

function TMemoX.Get_DragMode: TxDragMode;
begin
  Result := Ord(FDelphiControl.DragMode);
end;

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

function TMemoX.Get_Font: IFontDisp;
begin
  GetOleFont(FDelphiControl.Font, Result);
end;

function TMemoX.Get_HideSelection: WordBool;
begin
  Result := FDelphiControl.HideSelection;
end;

function TMemoX.Get_ImeMode: TxImeMode;
begin
  Result := Ord(FDelphiControl.ImeMode);
```

continues

LISTING 25.3 Continued

```
end;

function TMemoX.Get_ImeName: WideString;
begin
  Result := WideString(FDelphiControl.ImeName);
end;

function TMemoX.Get_MaxLength: Integer;
begin
  Result := FDelphiControl.MaxLength;
end;

function TMemoX.Get_Modified: WordBool;
begin
  Result := FDelphiControl.Modified;
end;

function TMemoX.Get_OEMConvert: WordBool;
begin
  Result := FDelphiControl.OEMConvert;
end;

function TMemoX.Get_ParentColor: WordBool;
begin
  Result := FDelphiControl.ParentColor;
end;

function TMemoX.Get_ParentCtl3D: WordBool;
begin
  Result := FDelphiControl.ParentCtl3D;
end;

function TMemoX.Get_ParentFont: WordBool;
begin
  Result := FDelphiControl.ParentFont;
end;

function TMemoX.Get_ReadOnly: WordBool;
begin
  Result := FDelphiControl.ReadOnly;
end;

function TMemoX.Get_ScrollBars: TxScrollStyle;
begin
  Result := Ord(FDelphiControl.ScrollBars);
```

```
end;

function TMemoX.Get_SelLength: Integer;
begin
  Result := FDelphiControl.SelLength;
end;

function TMemoX.Get_SelStart: Integer;
begin
  Result := FDelphiControl.SelStart;
end;

function TMemoX.Get_SelText: WideString;
begin
  Result := WideString(FDelphiControl.SelText);
end;

function TMemoX.Get_Text: WideString;
begin
  Result := WideString(FDelphiControl.Text);
end;

function TMemoX.Get_Visible: WordBool;
begin
  Result := FDelphiControl.Visible;
end;

function TMemoX.Get_WantReturns: WordBool;
begin
  Result := FDelphiControl.WantReturns;
end;

function TMemoX.Get_WantTabs: WordBool;
begin
  Result := FDelphiControl.WantTabs;
end;

function TMemoX.Get_WordWrap: WordBool;
begin
  Result := FDelphiControl.WordWrap;
end;

function TMemoX.GetControlsAlignment: TxAlignment;
begin
 Result := TxAlignment(FDelphiControl.GetControlsAlignment);
end;
```

continues

LISTING 25.3 Continued

```
function TMemoX.IsRightToLeft: WordBool;
begin
  Result := FDelphiControl.IsRightToLeft;
end;

function TMemoX.UseRightToLeftAlignment: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftAlignment;
end;

function TMemoX.UseRightToLeftReading: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftReading;
end;

function TMemoX.UseRightToLeftScrollBar: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftScrollBar;
end;

procedure TMemoX._Set_Font(const Value: IFontDisp);
begin
  SetOleFont(FDelphiControl.Font, Value);
end;

procedure TMemoX.AboutBox;
begin
  ShowMemoXAbout;
end;

procedure TMemoX.Clear;
begin
  FDelphiControl.Clear;
end;

procedure TMemoX.ClearSelection;
begin
  FDelphiControl.ClearSelection;
end;

procedure TMemoX.ClearUndo;
begin
  FDelphiControl.ClearUndo;
end;
```

```
procedure TMemoX.CopyToClipboard;
begin
  FDelphiControl.CopyToClipboard;
end;

procedure TMemoX.CutToClipboard;
begin
  FDelphiControl.CutToClipboard;
end;

procedure TMemoX.FlipChildren(AllLevels: WordBool);
begin
  FDelphiControl.FlipChildren(AllLevels);
end;

procedure TMemoX.InitiateAction;
begin
  FDelphiControl.InitiateAction;
end;

procedure TMemoX.PasteFromClipboard;
begin
  FDelphiControl.PasteFromClipboard;
end;

procedure TMemoX.SelectAll;
begin
  FDelphiControl.SelectAll;
end;

procedure TMemoX.Set_Alignment(Value: TxAlignment);
begin
  FDelphiControl.Alignment := TAlignment(Value);
end;

procedure TMemoX.Set_BiDiMode(Value: TxBiDiMode);
begin
  FDelphiControl.BiDiMode := TBiDiMode(Value);
end;

procedure TMemoX.Set_BorderStyle(Value: TxBorderStyle);
begin
  FDelphiControl.BorderStyle := TBorderStyle(Value);
end;

procedure TMemoX.Set_Color(Value: OLE_COLOR);
```

continues

LISTING 25.3 Continued

```
begin
  FDelphiControl.Color := TColor(Value);
end;

procedure TMemoX.Set_Ctl3D(Value: WordBool);
begin
  FDelphiControl.Ctl3D := Value;
end;

procedure TMemoX.Set_Cursor(Value: Smallint);
begin
  FDelphiControl.Cursor := TCursor(Value);
end;

procedure TMemoX.Set_DoubleBuffered(Value: WordBool);
begin
  FDelphiControl.DoubleBuffered := Value;
end;

procedure TMemoX.Set_DragCursor(Value: Smallint);
begin
  FDelphiControl.DragCursor := TCursor(Value);
end;

procedure TMemoX.Set_DragMode(Value: TxDragMode);
begin
  FDelphiControl.DragMode := TDragMode(Value);
end;

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

procedure TMemoX.Set_Font(var Value: IFontDisp);
begin
  SetOleFont(FDelphiControl.Font, Value);
end;

procedure TMemoX.Set_HideSelection(Value: WordBool);
begin
  FDelphiControl.HideSelection := Value;
end;

procedure TMemoX.Set_ImeMode(Value: TxImeMode);
```

```
begin
  FDelphiControl.ImeMode := TImeMode(Value);
end;

procedure TMemoX.Set_ImeName(const Value: WideString);
begin
  FDelphiControl.ImeName := TImeName(Value);
end;

procedure TMemoX.Set_MaxLength(Value: Integer);
begin
  FDelphiControl.MaxLength := Value;
end;

procedure TMemoX.Set_Modified(Value: WordBool);
begin
  FDelphiControl.Modified := Value;
end;

procedure TMemoX.Set_OEMConvert(Value: WordBool);
begin
  FDelphiControl.OEMConvert := Value;
end;

procedure TMemoX.Set_ParentColor(Value: WordBool);
begin
  FDelphiControl.ParentColor := Value;
end;

procedure TMemoX.Set_ParentCtl3D(Value: WordBool);
begin
  FDelphiControl.ParentCtl3D := Value;
end;

procedure TMemoX.Set_ParentFont(Value: WordBool);
begin
  FDelphiControl.ParentFont := Value;
end;

procedure TMemoX.Set_ReadOnly(Value: WordBool);
begin
  FDelphiControl.ReadOnly := Value;
end;

procedure TMemoX.Set_ScrollBars(Value: TxScrollStyle);
begin
```

continues

LISTING 25.3 Continued

```
  FDelphiControl.ScrollBars := TScrollStyle(Value);
end;

procedure TMemoX.Set_SelLength(Value: Integer);
begin
  FDelphiControl.SelLength := Value;
end;

procedure TMemoX.Set_SelStart(Value: Integer);
begin
  FDelphiControl.SelStart := Value;
end;

procedure TMemoX.Set_SelText(const Value: WideString);
begin
  FDelphiControl.SelText := String(Value);
end;

procedure TMemoX.Set_Text(const Value: WideString);
begin
  FDelphiControl.Text := TCaption(Value);
end;

procedure TMemoX.Set_Visible(Value: WordBool);
begin
  FDelphiControl.Visible := Value;
end;

procedure TMemoX.Set_WantReturns(Value: WordBool);
begin
  FDelphiControl.WantReturns := Value;
end;

procedure TMemoX.Set_WantTabs(Value: WordBool);
begin
  FDelphiControl.WantTabs := Value;
end;

procedure TMemoX.Set_WordWrap(Value: WordBool);
begin
  FDelphiControl.WordWrap := Value;
end;

procedure TMemoX.Undo;
```

```
begin
  FDelphiControl.Undo;
end;

procedure TMemoX.ChangeEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnChange;
end;

procedure TMemoX.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TMemoX.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TMemoX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

initialization
  TActiveXControlFactory.Create(ComServer, TMemoX, TMemo, Class_MemoX, 1, '',
    0, tmApartment);
end.
```

There is no doubt that Listings 25.1 through 25.3 contain a lot of code. Sometimes the sheer volume of code can make something appear daunting and difficult; however, if you look closely, you will see that no rocket science is involved in these files. What is pretty nifty is that you now have a fully functional ActiveX control (including an interface, a type library, and events) based on a memo control, and you have yet to write a line of code!

Note the helper functions that are used to convert back and forth between properties of `IStrings` and `IFont` to the native Delphi `TStrings` and `TFont` types. Each of these routines operates in a similar manner: They provide a bridge between an Object Pascal class and an Automation-compatible dispatch interface. Table 25.1 shows a list of VCL classes and their Automation interface equivalents.

TABLE 25.1 VCL Classes and their Corresponding Automation Interfaces

VCL Class	*Automation Interface*
`TFont`	`IFont`
`TPicture`	`IPicture`
`TStrings`	`IStrings`

NOTE

ActiveX defines the `IFont` and `IPicture` interfaces. However, the `IStrings` type is defined in VCL. Delphi provides a redistributable file named `StdVcl40.dll` that contains the type library that defines this interface. This library must be installed and registered on client machines in order for applications using an ActiveX control with `IStrings` properties to function properly.

The ActiveX Framework

The Delphi ActiveX framework (or *DAX*, for short) resides in the `AxCtrls` unit. An ActiveX control could be described as an Automation object on steroids, because it must implement the `IDispatch` interface (in addition to many others). Because of this fact, the DAX framework is similar to that of Automation objects, which you learned about in Chapter 23. `TActiveXControl` is a `TAutoObject` descendent that implements the interfaces required of an ActiveX control. The DAX framework works as a dual-object framework, where the ActiveX control portion contained in `TActiveXControl` communicates with a separate `TWinControl` class that contains the VCL control.

Like all COM objects, ActiveX controls are created from factories. DAX's `TActiveXControlFactory` serves as the factory for the `TActiveXControl` object. An instance of one of these factories is created in the `initialization` section of each control implementation file. The constructor for this class is defined as follows:

```
constructor TActiveXControlFactory.Create(ComServer: TComServerObject;
  ActiveXControlClass: TActiveXControlClass;
  WinControlClass: TWinControlClass; const ClassID: TGUID;
  ToolboxBitmapID: Integer; const LicStr: string; MiscStatus: Integer;
  ThreadingModel: TThreadingModel = tmSingle);
```

`ComServer` holds an instance of `TComServer`. Generally, the `ComServer` global declared in the `ComServ` unit is passed in this parameter.

`ActiveXControlClass` contains the name of the `TActiveXControl` descendant that is declared in the implementation file.

`WinControlClass` contains the name of the VCL `TWinControl` descendent that you want to encapsulate as an ActiveX control.

`ClassID` holds the CLSID of the control coclass as listed in the type library editor.

`ToolboxBitmapID` contains the resource identifier of the bitmap that should be used as the control's representation on the Component Palette.

`LicStr` holds the string that should be used as the control's license key string. If this is empty, the control is not licensed.

`MiscStatus` holds the `OLEMISC_`*`XXX`* status flags for the control. These flags are defined in the `ActiveX` unit. These `OLEMISC` flags are entered into the System Registry when the ActiveX control is registered. `OLEMISC` flags provide ActiveX control containers with information regarding various attributes of the ActiveX control. For example, there are `OLEMISC` flags that indicate how a control is painted and whether a control can contain other controls. These flags are fully documented on the Microsoft Developer's Network under the topic "`OLEMISC`."

Finally, `ThreadingModel` identifies the threading model that this control will be registered as supporting. It is important to note that setting this parameter to some particular threading model does not guarantee that your control is safer for that particular model; it only affects how the control is registered. Building in thread safety is up to you as the developer. See Chapter 23 for a discussion of each of the threading models.

Simple Frame Controls

One of the `OLEMISC_`*`XXX`* flags is `OLEMISC_SIMPLEFRAME`, which will automatically be added if `csAcceptsControls` is included in the VCL control's `ControlStyle` set. This makes the ActiveX control a simple frame control capable of containing other ActiveX controls in an ActiveX container application. The `TActiveXControl` class contains the necessary message-handling infrastructure to make simple frame controls work correctly. Occasionally, the wizard will add this flag to a control that you do not want to serve as a simple frame; in this case, it is okay to remove the flag from the class factory constructor call.

The Reflector Window

Some VCL controls require notification messages in order to properly function. For this purpose, DAX will create a reflector window whose job is to receive messages and forward them on to the VCL control. Standard VCL controls that require a reflector window will have the `csReflector` member included in their `ControlStyle` set. If you have a custom `TWinControl` that operates using notification messages, you should be sure to add this member to the `ControlStyle` set in the control's constructor.

Design Time Versus Runtime

VCL provides a simple means for determining whether a control is currently in design mode or run mode—by checking for the `csDesigning` member in the `ComponentState` set. Although you can to make this distinction for ActiveX controls, it is not so straightforward. It involves obtaining a pointer to the container's `IAmbientDispatch` dispinterface and checking the `UserMode` property on that dispinterface. You can use the following function for this purpose:

```
function IsControlRunning(Control: IUnknown): Boolean;
var
  OleObj: IOleObject;
  Site: IOleClientSite;
begin
  Result := True;
  // Get control's IOleObject pointer.  From that, get container's
  // IOleClientSite.  From that, get IAmbientDispatch.
  if (Control.QueryInterface(IOleObject, OleObj) = S_OK) and
    (OleObj.GetClientSite(Site) = S_OK) and (Site <> nil) then
    Result := (Site as IAmbientDispatch).UserMode;
end;
```

Control Licensing

We mentioned earlier in this chapter that the default DAX scheme for licensing involves an LIC file that should accompany the ActiveX control OCX file on development machines. As you saw earlier, the license string is one of the parameters to the ActiveX control's class factory constructor. When Make Control Licensed is selected in the wizard, this option will generate a GUID string that will be inserted into both the constructor call and the LIC file (you are free to modify the string later if you so choose). When the control is used at design time in a development tool, DAX will attempt to match the license string in the class factory with a string in the LIC file. If a match occurs, the control instance will be created. When an application that includes the licensed ActiveX control is compiled, the license string is embedded in the application, and the LIC file is not required to run the application.

The LIC file scheme for licensing is not the only one under the sun. For example, some developers find the use of an additional file cumbersome and prefer to store a license key in the Registry. Fortunately, DAX makes it very easy to implement an alternative licensing scheme such as this. The license check occurs in a `TActiveXControlFactory` method called `HasMachineLicense()`. By default, this method attempts to look up the licensing string in the LIC file, but you can have this method perform whatever check you want to determine licensing. For example, Listing 25.4 shows a `TActiveXControlFactory` descendent that looks in the Registry for the license key.

LISTING 25.4 An Alternative Scheme for Licensing

```
{ TRegLicAxControlFactory }

type
  TRegLicActiveXControlFactory = class(TActiveXControlFactory)
  protected
    function HasMachineLicense: Boolean; override;
  end;

function TRegLicActiveXControlFactory.HasMachineLicense: Boolean;
var
  Reg: TRegistry;
begin
  Result := True;
  if not SupportsLicensing then Exit;
  Reg := TRegistry.Create;
  try
    Reg.RootKey := HKEY_CLASSES_ROOT;
    // control is licensed if key is in registry
    Result := Reg.OpenKey('\Licenses\' + LicString, False);
  finally
    Reg.Free;
  end;
end;
```

A Registry file (REG) can be used to place the license key in the Registry on a licensed machine. This is shown in Listing 25.5.

LISTING 25.5 The Licensing REG File

```
REGEDIT4

[HKEY_CLASSES_ROOT\Licenses\{C06EFEA0-06B2-11D1-A9BF-B18A9F703311}]
@= "Licensing info for DDG demo ActiveX control"
```

Property Pages

Property pages provide a means for modifying the properties of an ActiveX control through a custom dialog. A control's property pages are added as pages in a tabbed dialog that is created by ActiveX. Property page dialogs are usually invoked from a local right-click menu provided by the control's host container.

Standard Property Pages

DAX provides standard property pages for properties of type `IStrings`, `IPicture`, `TColor`, and `IFont`. The CLSIDs for these property pages are found in the `AxCtrls` unit. They are declared as follows:

```
const
  { Delphi property page CLSIDs }
  Class_DColorPropPage: TGUID = '{5CFF5D59-5946-11D0-BDEF-00A024D1875C}';
  Class_DFontPropPage: TGUID = '{5CFF5D5B-5946-11D0-BDEF-00A024D1875C}';
  Class_DPicturePropPage: TGUID = '{5CFF5D5A-5946-11D0-BDEF-00A024D1875C}';
  Class_DStringPropPage: TGUID = '{F42D677E-754B-11D0-BDFB-00A024D1875C}';
```

Using any of these property pages in your control is a simple matter: Just pass one of these CLSIDs to the `DefinePropertyPage()` procedural parameter in the `DefinePropertyPages()` method of your ActiveX control, as shown here:

```
procedure TMemoX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  DefinePropertyPage(Class_DColorPropPage);
  DefinePropertyPage(Class_DFontPropPage);
  DefinePropertyPage(Class_DStringPropPage);
end;
```

Figures 25.4 through 25.7 show each of the standard DAX property pages.

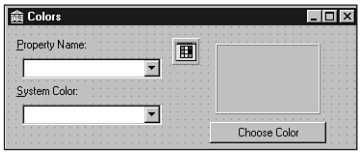

FIGURE 25.4
DAX Colors property page.

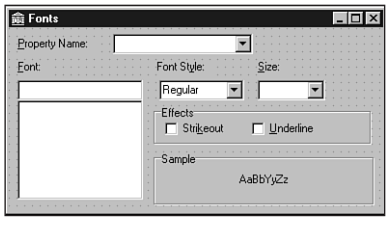

FIGURE 25.5
DAX Fonts property page.

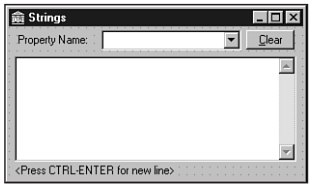

FIGURE 25.6
DAX Strings property page.

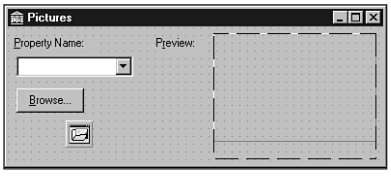

FIGURE 25.7
DAX Pictures property page.

Each of these property pages operates similarly: The combo box contains the names of each of the properties of the specified type. You just select the property name, set the value in the dialog, and then click OK to modify the selected property.

NOTE

If you want to use the standard DAX property pages, you must distribute `StdVcl40.dll` along with your OCX file. As mentioned earlier in this chapter, this file contains the definition for `IStrings` as well as the `IProvider` and `IDataBroker` interfaces. Additionally, `StdVcl40.dll` contains the implementation for each of the DAX property pages. You must also ensure that both the OCX file and `StdVcl40.dll` have been registered on the target machine.

Custom Property Pages

To help illustrate the creation of custom property pages, we will create a control that is more interesting than the simple `Memo` control we have been working with so far. Listing 25.6 shows the implementation file for the `TCardX` ActiveX control. This control is an encapsulation of the playing card VCL control that comes from the `Cards` unit, which you will find in the `\Code\Comps` subdirectory of the CD-ROM accompanying this book.

LISTING 25.6 CardImpl.pas: Implementation File for the TCardX ActiveX Control

```
unit CardImpl;

interface

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

type
  TCardX = class(TActiveXControl, ICardX)
  private
    { Private declarations }
    FDelphiControl: TCard;
    FEvents: ICardXEvents;
    procedure ClickEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
  protected
    { Protected declarations }
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
      override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    procedure InitializeControl; override;
    function ClassNameIs(const Name: WideString): WordBool; safecall;
    function DrawTextBiDiModeFlags(Flags: Integer): Integer; safecall;
    function DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
    function Get_BackColor: OLE_COLOR; safecall;
    function Get_BiDiMode: TxBiDiMode; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_Cursor: Smallint; safecall;
    function Get_DoubleBuffered: WordBool; safecall;
    function Get_DragCursor: Smallint; safecall;
    function Get_DragMode: TxDragMode; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_FaceUp: WordBool; safecall;
    function Get_ParentColor: WordBool; safecall;
    function Get_Suit: TxCardSuit; safecall;
    function Get_Value: TxCardValue; safecall;
    function Get_Visible: WordBool; safecall;
    function GetControlsAlignment: TxAlignment; safecall;
    function IsRightToLeft: WordBool; safecall;
    function UseRightToLeftAlignment: WordBool; safecall;
    function UseRightToLeftReading: WordBool; safecall;
    function UseRightToLeftScrollBar: WordBool; safecall;
    procedure FlipChildren(AllLevels: WordBool); safecall;
```

```
    procedure InitiateAction; safecall;
    procedure Set_BackColor(Value: OLE_COLOR); safecall;
    procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    procedure Set_DragMode(Value: TxDragMode); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_FaceUp(Value: WordBool); safecall;
    procedure Set_ParentColor(Value: WordBool); safecall;
    procedure Set_Suit(Value: TxCardSuit); safecall;
    procedure Set_Value(Value: TxCardValue); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    procedure AboutBox; safecall;
  end;

implementation

uses ComObj, About, CardPP;

{ TCardX }

procedure TCardX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  DefinePropertyPage(Class_DColorPropPage);
  DefinePropertyPage(Class_CardPropPage);
end;

procedure TCardX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as ICardXEvents;
end;

procedure TCardX.InitializeControl;
begin
  FDelphiControl := Control as TCard;
  FDelphiControl.OnClick := ClickEvent;
  FDelphiControl.OnDblClick := DblClickEvent;
  FDelphiControl.OnKeyPress := KeyPressEvent;
end;

function TCardX.ClassNameIs(const Name: WideString): WordBool;
begin
  Result := FDelphiControl.ClassNameIs(Name);
end;
```

continues

LISTING 25.6 Continued

```
function TCardX.DrawTextBiDiModeFlags(Flags: Integer): Integer;
begin
  Result := FDelphiControl.DrawTextBiDiModeFlags(Flags);
end;

function TCardX.DrawTextBiDiModeFlagsReadingOnly: Integer;
begin
  Result := FDelphiControl.DrawTextBiDiModeFlagsReadingOnly;
end;

function TCardX.Get_BackColor: OLE_COLOR;
begin
  Result := OLE_COLOR(FDelphiControl.BackColor);
end;

function TCardX.Get_BiDiMode: TxBiDiMode;
begin
  Result := Ord(FDelphiControl.BiDiMode);
end;

function TCardX.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(FDelphiControl.Color);
end;

function TCardX.Get_Cursor: Smallint;
begin
  Result := Smallint(FDelphiControl.Cursor);
end;

function TCardX.Get_DoubleBuffered: WordBool;
begin
  Result := FDelphiControl.DoubleBuffered;
end;

function TCardX.Get_DragCursor: Smallint;
begin
  Result := Smallint(FDelphiControl.DragCursor);
end;

function TCardX.Get_DragMode: TxDragMode;
begin
  Result := Ord(FDelphiControl.DragMode);
end;
```

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

function TCardX.Get_FaceUp: WordBool;
begin
  Result := FDelphiControl.FaceUp;
end;

function TCardX.Get_ParentColor: WordBool;
begin
  Result := FDelphiControl.ParentColor;
end;

function TCardX.Get_Suit: TxCardSuit;
begin
  Result := Ord(FDelphiControl.Suit);
end;

function TCardX.Get_Value: TxCardValue;
begin
  Result := Ord(FDelphiControl.Value);
end;

function TCardX.Get_Visible: WordBool;
begin
  Result := FDelphiControl.Visible;
end;

function TCardX.GetControlsAlignment: TxAlignment;
begin
 Result := TxAlignment(FDelphiControl.GetControlsAlignment);
end;

function TCardX.IsRightToLeft: WordBool;
begin
  Result := FDelphiControl.IsRightToLeft;
end;

function TCardX.UseRightToLeftAlignment: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftAlignment;
end;

function TCardX.UseRightToLeftReading: WordBool;
```

continues

LISTING 25.6 Continued

```
begin
  Result := FDelphiControl.UseRightToLeftReading;
end;

function TCardX.UseRightToLeftScrollBar: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftScrollBar;
end;

procedure TCardX.FlipChildren(AllLevels: WordBool);
begin
  FDelphiControl.FlipChildren(AllLevels);
end;

procedure TCardX.InitiateAction;
begin
  FDelphiControl.InitiateAction;
end;

procedure TCardX.Set_BackColor(Value: OLE_COLOR);
begin
  FDelphiControl.BackColor := TColor(Value);
end;

procedure TCardX.Set_BiDiMode(Value: TxBiDiMode);
begin
  FDelphiControl.BiDiMode := TBiDiMode(Value);
end;

procedure TCardX.Set_Color(Value: OLE_COLOR);
begin
  FDelphiControl.Color := TColor(Value);
end;

procedure TCardX.Set_Cursor(Value: Smallint);
begin
  FDelphiControl.Cursor := TCursor(Value);
end;

procedure TCardX.Set_DoubleBuffered(Value: WordBool);
begin
  FDelphiControl.DoubleBuffered := Value;
end;

procedure TCardX.Set_DragCursor(Value: Smallint);
```

```
begin
  FDelphiControl.DragCursor := TCursor(Value);
end;

procedure TCardX.Set_DragMode(Value: TxDragMode);
begin
  FDelphiControl.DragMode := TDragMode(Value);
end;

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

procedure TCardX.Set_FaceUp(Value: WordBool);
begin
  FDelphiControl.FaceUp := Value;
end;

procedure TCardX.Set_ParentColor(Value: WordBool);
begin
  FDelphiControl.ParentColor := Value;
end;

procedure TCardX.Set_Suit(Value: TxCardSuit);
begin
  FDelphiControl.Suit := TCardSuit(Value);
end;

procedure TCardX.Set_Value(Value: TxCardValue);
begin
  FDelphiControl.Value := TCardValue(Value);
end;

procedure TCardX.Set_Visible(Value: WordBool);
begin
  FDelphiControl.Visible := Value;
end;

procedure TCardX.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TCardX.DblClickEvent(Sender: TObject);
begin
```

continues

LISTING 25.6 Continued

```
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TCardX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TCardX.AboutBox;
begin
  ShowCardXAbout;
end;

initialization
  TActiveXControlFactory.Create(ComServer, TCardX, TCard, Class_CardX,
    1, '', 0, tmApartment);
end.
```

This unit is essentially what was generated by the wizard, except for the two lines of code shown in the `DefinePropertyPages()` method. In this method, you can see that we employ the standard VCL Color property page in addition to a custom property page whose CLSID is defined as `Class_CardPropPage`. This property page was created by selecting the Property Page item from the ActiveX page of the New Items dialog. Figure 25.8 shows this property page in the Form Designer, and Listing 25.7 shows the source code for this property page.

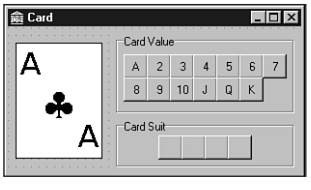

FIGURE 25.8
A property page in the Form Designer.

LISTING 25.7 The Property Page Unit: `CardPP.pas`

```
unit CardPP;

interface

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

type
  TCardPropPage = class(TPropertyPage)
    Card1: TCard;
    ValueGroup: TGroupBox;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    SpeedButton11: TSpeedButton;
    SpeedButton12: TSpeedButton;
    SuitGroup: TGroupBox;
    SpeedButton13: TSpeedButton;
    SpeedButton14: TSpeedButton;
    SpeedButton15: TSpeedButton;
    SpeedButton16: TSpeedButton;
    SpeedButton17: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
  protected
    procedure UpdatePropertyPage; override;
    procedure UpdateObject; override;
  end;

const
  Class_CardPropPage: TGUID = '{C06EFEA1-06B2-11D1-A9BF-B18A9F703311}';

implementation

{$R *.DFM}
```

continues

LISTING 25.7 Continued

```
procedure TCardPropPage.UpdatePropertyPage;
var
  i: Integer;
  AValue, ASuit: Integer;
begin
  // get suit and value
  AValue := OleObject.Value;
  ASuit := OleObject.Suit;
  // set card correctly
  Card1.Value := TCardValue(AValue);
  Card1.Suit := TCardSuit(ASuit);
  // set correct value speedbutton
  with ValueGroup do
    for i := 0 to ControlCount - 1 do
      if (Controls[i] is TSpeedButton) and
        (TSpeedButton(Controls[i]).Tag = AValue) then
        TSpeedButton(Controls[i]).Down := True;
  // set correct suit speedbutton
  with SuitGroup do
    for i := 0 to ControlCount - 1 do
      if (Controls[i] is TSpeedButton) and
        (TSpeedButton(Controls[i]).Tag = ASuit) then
        TSpeedButton(Controls[i]).Down := True;
end;

procedure TCardPropPage.UpdateObject;
var
  i: Integer;
begin
  // set correct value speedbutton
  with ValueGroup do
    for i := 0 to ControlCount - 1 do
      if (Controls[i] is TSpeedButton) and TSpeedButton(Controls[i]).Down then
      begin
        OleObject.Value := TSpeedButton(Controls[i]).Tag;
        Break;
      end;
  // set correct suit speedbutton
  with SuitGroup do
    for i := 0 to ControlCount - 1 do
      if (Controls[i] is TSpeedButton) and TSpeedButton(Controls[i]).Down then
      begin
        OleObject.Suit := TSpeedButton(Controls[i]).Tag;
        Break;
      end;
end;
```

```
procedure TCardPropPage.FormCreate(Sender: TObject);
const
  // ordinal values of "suit" characters in Symbol font:
  SSuits: PChar = #167#168#169#170;
var
  i: Integer;
begin
  // set up captions of suit speedbuttons using high
  // characters in Symbol font
  with SuitGroup do
    for i := 0 to ControlCount - 1 do
      if Controls[i] is TSpeedButton then
        TSpeedButton(Controls[i]).Caption := SSuits[i];
end;

procedure TCardPropPage.SpeedButton1Click(Sender: TObject);
begin
  if Sender is TSpeedButton then
  begin
    with TSpeedButton(Sender) do
    begin
      if Parent = ValueGroup then
        Card1.Value := TCardValue(Tag)
      else if Parent = SuitGroup then
        Card1.Suit := TCardSuit(Tag);
    end;
    Modified;
  end;
end;

initialization
  TActiveXPropertyPageFactory.Create(
    ComServer,
    TCardPropPage,
    Class_CardPropPage);
end.
```

You must communicate with the ActiveX control from the property page using its `OleObject` field. `OleObject` is a variant that holds a reference to the control's `IDispatch` interface. The `UpdatePropertyPage()` and `UpdateObject()` methods are generated by the wizard. `UpdatePropertyPage()` is called when the property page is invoked. In this method, you must set the contents of the page to match the current values of the ActiveX control as indicated in the `OleObject` property. `UpdateObject()` will be called when the user clicks the OK or Apply button in the Property Page dialog. In this method, you should use the `OleObject` property to set the ActiveX control properties to those indicated by the property page.

In this example, the property page allows you to edit the suit or value of the `TCardX` ActiveX control. As you modify the suit or value using speedbuttons in the dialog, a `TCard` VCL control residing on the property page is changed to reflect the current suit and value. Notice also that when a speedbutton is clicked, the property page's `Modified()` procedure is called to set the modified flag of the Property Page dialog. This enables the Apply button on the dialog.

This property page is shown in action in Figure 25.9.

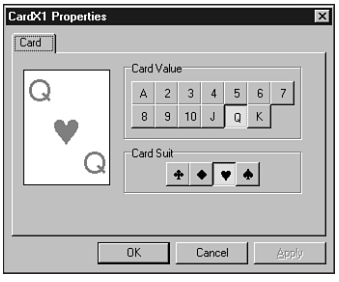

FIGURE 25.9
The Card property page in action.

ActiveForms

Functionally, ActiveForms work very much the same as the ActiveX controls you learned about earlier in this chapter. The primary difference is that the VCL control upon which you base an ActiveX control does not really change after you run the wizard, whereas the whole point of an ActiveForm is that it changes constantly as it is manipulated in the designer. Because the ActiveForm's wizard and framework are essentially the same as the ones for ActiveX controls, we will not rehash that material. Instead, let's focus on some interesting things you can do with ActiveForms.

Adding Properties to ActiveForms

One problem with ActiveForms is that their representation in the type library consists of "flat" interfaces rather than the nested components you are familiar with in VCL. This means that if you have a form with several buttons, they cannot easily be addressed in the VCL manner of *`ActiveForm.Button.ButtonProperty`* as an ActiveForm. Instead, the easiest way to accomplish this is to surface the button properties in question as properties of the ActiveForm itself. The DAX framework makes adding properties to ActiveForms a pretty painless process; you just need to follow a couple steps. Here is what's required to publish the `Caption` property of a button on an ActiveForm:

1. Add a new published property to the ActiveForm declaration in the implementation file. This property will be called `ButtonCaption`, and it will have reader and writer methods that modify the `Caption` property of the button.
2. Add a new property of the same name to the ActiveForm's interface in the type library. Delphi will automatically write the skeletons for the reader and writer methods for this property, and you must implement them by reading and writing the ActiveForm's `ButtonCaption` property.

The implementation file for this component is shown in Listing 25.8.

LISTING 25.8 Adding Properties to ActiveForms

```
unit AFImpl;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, AFrm_TLB, StdCtrls;

type
  TActiveFormX = class(TActiveForm, IActiveFormX)
    Button1: TButton;
  private
    { Private declarations }
    FEvents: IActiveFormXEvents;
    procedure ActivateEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure CreateEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure DeactivateEvent(Sender: TObject);
    procedure DestroyEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
    procedure PaintEvent(Sender: TObject);
    function GetButtonCaption: string;
    procedure SetButtonCaption(const Value: string);
  protected
    { Protected declarations }
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
      override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    function Get_Active: WordBool; safecall;
    function Get_AutoScroll: WordBool; safecall;
    function Get_AutoSize: WordBool; safecall;
    function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
```

continues

LISTING 25.8 Continued

```
    function Get_BiDiMode: TxBiDiMode; safecall;
    function Get_Caption: WideString; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_Cursor: Smallint; safecall;
    function Get_DoubleBuffered: WordBool; safecall;
    function Get_DropTarget: WordBool; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_Font: IFontDisp; safecall;
    function Get_HelpFile: WideString; safecall;
    function Get_KeyPreview: WordBool; safecall;
    function Get_PixelsPerInch: Integer; safecall;
    function Get_PrintScale: TxPrintScale; safecall;
    function Get_Scaled: WordBool; safecall;
    function Get_Visible: WordBool; safecall;
    procedure _Set_Font(const Value: IFontDisp); safecall;
    procedure AboutBox; safecall;
    procedure Set_AutoScroll(Value: WordBool); safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
    procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
    procedure Set_Caption(const Value: WideString); safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DropTarget(Value: WordBool); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(var Value: IFontDisp); safecall;
    procedure Set_HelpFile(const Value: WideString); safecall;
    procedure Set_KeyPreview(Value: WordBool); safecall;
    procedure Set_PixelsPerInch(Value: Integer); safecall;
    procedure Set_PrintScale(Value: TxPrintScale); safecall;
    procedure Set_Scaled(Value: WordBool); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    function Get_ButtonCaption: WideString; safecall;
    procedure Set_ButtonCaption(const Value: WideString); safecall;
  public
    { Public declarations }
    procedure Initialize; override;
  published
    property ButtonCaption: string read GetButtonCaption
      write SetButtonCaption;
  end;

implementation
```

```
uses ComObj, ComServ, About1;

{$R *.DFM}

{ TActiveFormX }

procedure TActiveFormX.DefinePropertyPages(DefinePropertyPage:
  TDefinePropertyPage);
begin
  { Define property pages here.  Property pages are defined by calling
    DefinePropertyPage with the class id of the page.  For example,
      DefinePropertyPage(Class_ActiveFormXPage); }
end;

procedure TActiveFormX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IActiveFormXEvents;
end;

procedure TActiveFormX.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
end;

function TActiveFormX.Get_Active: WordBool;
begin
  Result := Active;
end;

function TActiveFormX.Get_AutoScroll: WordBool;
begin
  Result := AutoScroll;
end;

function TActiveFormX.Get_AutoSize: WordBool;
begin
  Result := AutoSize;
end;
```

continues

LISTING 25.8 Continued

```
function TActiveFormX.Get_AxBorderStyle: TxActiveFormBorderStyle;
begin
  Result := Ord(AxBorderStyle);
end;

function TActiveFormX.Get_BiDiMode: TxBiDiMode;
begin
  Result := Ord(BiDiMode);
end;

function TActiveFormX.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

function TActiveFormX.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(Color);
end;

function TActiveFormX.Get_Cursor: Smallint;
begin
  Result := Smallint(Cursor);
end;

function TActiveFormX.Get_DoubleBuffered: WordBool;
begin
  Result := DoubleBuffered;
end;

function TActiveFormX.Get_DropTarget: WordBool;
begin
  Result := DropTarget;
end;

function TActiveFormX.Get_Enabled: WordBool;
begin
  Result := Enabled;
end;

function TActiveFormX.Get_Font: IFontDisp;
begin
  GetOleFont(Font, Result);
end;
```

```
function TActiveFormX.Get_HelpFile: WideString;
begin
  Result := WideString(HelpFile);
end;

function TActiveFormX.Get_KeyPreview: WordBool;
begin
  Result := KeyPreview;
end;

function TActiveFormX.Get_PixelsPerInch: Integer;
begin
  Result := PixelsPerInch;
end;

function TActiveFormX.Get_PrintScale: TxPrintScale;
begin
  Result := Ord(PrintScale);
end;

function TActiveFormX.Get_Scaled: WordBool;
begin
  Result := Scaled;
end;

function TActiveFormX.Get_Visible: WordBool;
begin
  Result := Visible;
end;

procedure TActiveFormX._Set_Font(const Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;

procedure TActiveFormX.AboutBox;
begin
  ShowActiveFormXAbout;
end;

procedure TActiveFormX.Set_AutoScroll(Value: WordBool);
begin
  AutoScroll := Value;
end;

procedure TActiveFormX.Set_AutoSize(Value: WordBool);
```

continues

LISTING 25.8 Continued

```
begin
  AutoSize := Value;
end;

procedure TActiveFormX.Set_AxBorderStyle(Value: TxActiveFormBorderStyle);
begin
  AxBorderStyle := TActiveFormBorderStyle(Value);
end;

procedure TActiveFormX.Set_BiDiMode(Value: TxBiDiMode);
begin
  BiDiMode := TBiDiMode(Value);
end;

procedure TActiveFormX.Set_Caption(const Value: WideString);
begin
  Caption := TCaption(Value);
end;

procedure TActiveFormX.Set_Color(Value: OLE_COLOR);
begin
  Color := TColor(Value);
end;

procedure TActiveFormX.Set_Cursor(Value: Smallint);
begin
  Cursor := TCursor(Value);
end;

procedure TActiveFormX.Set_DoubleBuffered(Value: WordBool);
begin
  DoubleBuffered := Value;
end;

procedure TActiveFormX.Set_DropTarget(Value: WordBool);
begin
  DropTarget := Value;
end;

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

procedure TActiveFormX.Set_Font(var Value: IFontDisp);
```

```
begin
  SetOleFont(Font, Value);
end;

procedure TActiveFormX.Set_HelpFile(const Value: WideString);
begin
  HelpFile := String(Value);
end;

procedure TActiveFormX.Set_KeyPreview(Value: WordBool);
begin
  KeyPreview := Value;
end;

procedure TActiveFormX.Set_PixelsPerInch(Value: Integer);
begin
  PixelsPerInch := Value;
end;

procedure TActiveFormX.Set_PrintScale(Value: TxPrintScale);
begin
  PrintScale := TPrintScale(Value);
end;

procedure TActiveFormX.Set_Scaled(Value: WordBool);
begin
  Scaled := Value;
end;

procedure TActiveFormX.Set_Visible(Value: WordBool);
begin
  Visible := Value;
end;

procedure TActiveFormX.ActivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnActivate;
end;

procedure TActiveFormX.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TActiveFormX.CreateEvent(Sender: TObject);
begin
```

continues

LISTING 25.8 Continued

```
  if FEvents <> nil then FEvents.OnCreate;
end;

procedure TActiveFormX.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TActiveFormX.DeactivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDeactivate;
end;

procedure TActiveFormX.DestroyEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDestroy;
end;

procedure TActiveFormX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TActiveFormX.PaintEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnPaint;
end;

function TActiveFormX.GetButtonCaption: string;
begin
  Result := Button1.Caption;
end;

procedure TActiveFormX.SetButtonCaption(const Value: string);
begin
  Button1.Caption := Value;
end;

function TActiveFormX.Get_ButtonCaption: WideString;
begin
  Result := ButtonCaption;
```

```
end;

procedure TActiveFormX.Set_ButtonCaption(const Value: WideString);
begin
  ButtonCaption := Value;
end;

initialization
  TActiveFormFactory.Create(ComServer, TActiveFormControl, TActiveFormX,
    Class_ActiveFormX, 1, '', OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
    tmApartment);
end.
```

ActiveX on the Web

An ideal use for ActiveForms is as a vehicle for delivering small applications over the World Wide Web. Smaller ActiveX controls are also useful for enhancing the appearance and usefulness of Web pages. However, in order to get the most out of Delphi-written ActiveX controls on the Web, you need to know a few things about control streaming, safety, and communication with the browser.

Communicating with the Web Browser

Because ActiveX controls can run within the context of a Web browser, it makes sense that Web browsers expose functions and interfaces that allow ActiveX controls to manipulate them. Most of these functions and interfaces are located in the `UrlMon` unit (that's Jamaican Web talk). Among the simplest of these functions are the `Hlink`*`XXX`*`()` functions, which cause the browser to hyperlink to different locations. For example, the `HlinkGoForward()` and `HlinkGoBack()` functions cause the browser to travel forward or back in its location stack. The `HlinkNavigateString()` function causes the browser to travel to a specified URL. These functions are defined in `UrlMon` as follows:

```
function HlinkGoBack(pUnk: IUnknown): HResult; stdcall;
function HlinkGoForward(pUnk: IUnknown): HResult; stdcall;
function HlinkNavigateString(pUnk: IUnknown; szTarget: PWideChar): HResult;
  stdcall;
```

The `pUnk` parameter for each of these functions is the `IUnknown` interface for the ActiveX control. In the case of ActiveX controls, you can pass *`Control`* `as IUnknown` in this parameter. In the case of ActiveForms, you should pass `IUnknown(VclComObject)` in this parameter. The `szTarget` parameter of `HlinkNavigateString()` represents the URL you want to use.

A more ambitious task would be to use the `URLDownloadToFile()` function to download a file from the server to the local machine. This method is defined in `UrlMon` as follows:

```
function URLDownloadToFile(p1: IUnknown; p2: PChar; p3: PChar; p4: DWORD;
  p5: IBindStatusCallback): HResult; stdcall;
```

Helpful parameter names, eh? `p1` represents the `IUnknown` interface for the ActiveX control, similar to the `pUnk` parameter of the `HlinkXXX()` functions. `p2` holds the URL of the file to be downloaded. `p3` is the name of the local file that will be filled with the data of the file specified by `p2`. `p4` must be set to `0`, and `p5` holds an optional `IBindStatusCallback` interface pointer. This interface can be used to obtain incremental information on the file as it downloads.

Listing 25.9 shows the implementation file for an ActiveForm that implements these methods. It also demonstrates a simple example of implementing the `IBindStatusCallback` interface.

LISTING 25.9 An ActiveForm that Uses `UrlMon` Functions

```
unit UrlTestMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, UrlTest_TLB, UrlMon, StdCtrls, MPlayer, ExtCtrls,
  ComCtrls;

type
  TUrlTestForm = class(TActiveForm, IUrlTestForm, IBindStatusCallback)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    MediaPlayer1: TMediaPlayer;
    Panel1: TPanel;
    Button1: TButton;
    StatusPanel: TPanel;
    ProgressBar1: TProgressBar;
    ServerName: TEdit;
    StaticText1: TStaticText;
    procedure Label1Click(Sender: TObject);
    procedure Label2Click(Sender: TObject);
    procedure Label3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    FEvents: IUrlTestFormEvents;
    procedure ActivateEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure CreateEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
```

```
  procedure DeactivateEvent(Sender: TObject);
  procedure DestroyEvent(Sender: TObject);
  procedure KeyPressEvent(Sender: TObject; var Key: Char);
  procedure PaintEvent(Sender: TObject);
protected
  { IBindStatusCallback }
  function OnStartBinding(dwReserved: DWORD; pib: IBinding): HResult;
    stdcall;
  function GetPriority(out nPriority): HResult; stdcall;
  function OnLowResource(reserved: DWORD): HResult; stdcall;
  function OnProgress(ulProgress, ulProgressMax, ulStatusCode: ULONG;
    szStatusText: LPCWSTR): HResult; stdcall;
  function OnStopBinding( hRes: HResult; szError: PWideChar ): HResult;
    stdcall;
  function GetBindInfo(out grfBINDF: DWORD; var bindinfo: TBindInfo):
    HResult;
    stdcall;
  function OnDataAvailable(grfBSCF: DWORD; dwSize: DWORD;
    formatetc: PFormatEtc; stgmed: PStgMedium): HResult; stdcall;
  function OnObjectAvailable(const iid: TGUID; punk: IUnknown): HResult;
    stdcall;
  { UrlTestForm }
  procedure EventSinkChanged(const EventSink: IUnknown); override;
  procedure Initialize; override;
  function Get_Active: WordBool; safecall;
  function Get_AutoScroll: WordBool; safecall;
  function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
  function Get_Caption: WideString; safecall;
  function Get_Color: OLE_COLOR; safecall;
  function Get_Cursor: Smallint; safecall;
  function Get_DropTarget: WordBool; safecall;
  function Get_Enabled: WordBool; safecall;
  function Get_Font: IFontDisp; safecall;
  function Get_HelpFile: WideString; safecall;
  function Get_KeyPreview: WordBool; safecall;
  function Get_PixelsPerInch: Integer; safecall;
  function Get_PrintScale: TxPrintScale; safecall;
  function Get_Scaled: WordBool; safecall;
  function Get_Visible: WordBool; safecall;
  function Get_WindowState: TxWindowState; safecall;
  procedure Set_AutoScroll(Value: WordBool); safecall;
  procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
  procedure Set_Caption(const Value: WideString); safecall;
  procedure Set_Color(Color: OLE_COLOR); safecall;
  procedure Set_Cursor(Value: Smallint); safecall;
  procedure Set_DropTarget(Value: WordBool); safecall;
```

continues

LISTING 25.9 Continued

```
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(const Font: IFontDisp); safecall;
    procedure Set_HelpFile(const Value: WideString); safecall;
    procedure Set_KeyPreview(Value: WordBool); safecall;
    procedure Set_PixelsPerInch(Value: Integer); safecall;
    procedure Set_PrintScale(Value: TxPrintScale); safecall;
    procedure Set_Scaled(Value: WordBool); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    procedure Set_WindowState(Value: TxWindowState); safecall;
  public
    { Public declarations }
  end;

implementation

uses ComObj, ComServ;

{$R *.DFM}

{ TUrlTestForm.IBindStatusCallback }

function TUrlTestForm.OnStartBinding(dwReserved: DWORD; pib: IBinding):
  HResult;
begin
  Result := S_OK;
end;

function TUrlTestForm.GetPriority(out nPriority): HResult;
begin
  HRESULT(Result) := S_OK;
end;

function TUrlTestForm.OnLowResource(reserved: DWORD): HResult;
begin
  Result := S_OK;
end;

function TUrlTestForm.OnProgress(ulProgress, ulProgressMax, ulStatusCode:
  ULONG;
  szStatusText: LPCWSTR): HResult; stdcall;
begin
  Result := S_OK;
  ProgressBar1.Max := ulProgressMax;
  ProgressBar1.Position := ulProgress;
  StatusPanel.Caption := szStatusText;
```

```
end;

function TUrlTestForm.OnStopBinding(hRes: HResult; szError: PWideChar ):
  HResult;
begin
  Result := S_OK;
  if hRes = S_OK then
  begin
    MediaPlayer1.FileName := 'c:\temp\testavi.avi';
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;
end;

function TUrlTestForm.GetBindInfo(out grfBINDF: DWORD; var bindinfo:
  TBindInfo):
  HResult; stdcall;
begin
  Result := S_OK;
end;

function TUrlTestForm.OnDataAvailable(grfBSCF: DWORD; dwSize: DWORD;
  formatetc: PFormatEtc; stgmed: PStgMedium): HResult; stdcall;
begin
  Result := S_OK;
end;

function TUrlTestForm.OnObjectAvailable(const iid: TGUID; punk: IUnknown):
  HResult; stdcall;
begin
  Result := S_OK;
end;

{ TUrlTestForm }

procedure TUrlTestForm.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IUrlTestFormEvents;
end;

procedure TUrlTestForm.Initialize;
begin
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
```

continues

LISTING 25.9 Continued

```
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
end;

function TUrlTestForm.Get_Active: WordBool;
begin
  Result := Active;
end;

function TUrlTestForm.Get_AutoScroll: WordBool;
begin
  Result := AutoScroll;
end;

function TUrlTestForm.Get_AxBorderStyle: TxActiveFormBorderStyle;
begin
  Result := Ord(AxBorderStyle);
end;

function TUrlTestForm.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

function TUrlTestForm.Get_Color: OLE_COLOR;
begin
  Result := Color;
end;

function TUrlTestForm.Get_Cursor: Smallint;
begin
  Result := Smallint(Cursor);
end;

function TUrlTestForm.Get_DropTarget: WordBool;
begin
  Result := DropTarget;
end;

function TUrlTestForm.Get_Enabled: WordBool;
begin
  Result := Enabled;
end;
```

```
function TUrlTestForm.Get_Font: IFontDisp;
begin
  GetOleFont(Font, Result);
end;

function TUrlTestForm.Get_HelpFile: WideString;
begin
  Result := WideString(HelpFile);
end;

function TUrlTestForm.Get_KeyPreview: WordBool;
begin
  Result := KeyPreview;
end;

function TUrlTestForm.Get_PixelsPerInch: Integer;
begin
  Result := PixelsPerInch;
end;

function TUrlTestForm.Get_PrintScale: TxPrintScale;
begin
  Result := Ord(PrintScale);
end;

function TUrlTestForm.Get_Scaled: WordBool;
begin
  Result := Scaled;
end;

function TUrlTestForm.Get_Visible: WordBool;
begin
  Result := Visible;
end;

function TUrlTestForm.Get_WindowState: TxWindowState;
begin
  Result := Ord(WindowState);
end;

procedure TUrlTestForm.Set_AutoScroll(Value: WordBool);
begin
  AutoScroll := Value;
end;

procedure TUrlTestForm.Set_AxBorderStyle(Value: TxActiveFormBorderStyle);
```

continues

LISTING 25.9 Continued

```
begin
  AxBorderStyle := TActiveFormBorderStyle(Value);
end;

procedure TUrlTestForm.Set_Caption(const Value: WideString);
begin
  Caption := TCaption(Value);
end;

procedure TUrlTestForm.Set_Color(Color: OLE_COLOR);
begin
  Self.Color := Color;
end;

procedure TUrlTestForm.Set_Cursor(Value: Smallint);
begin
  Cursor := TCursor(Value);
end;

procedure TUrlTestForm.Set_DropTarget(Value: WordBool);
begin
  DropTarget := Value;
end;

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

procedure TUrlTestForm.Set_Font(const Font: IFontDisp);
begin
  SetOleFont(Self.Font, Font);
end;

procedure TUrlTestForm.Set_HelpFile(const Value: WideString);
begin
  HelpFile := String(Value);
end;

procedure TUrlTestForm.Set_KeyPreview(Value: WordBool);
begin
  KeyPreview := Value;
end;

procedure TUrlTestForm.Set_PixelsPerInch(Value: Integer);
```

```
begin
  PixelsPerInch := Value;
end;

procedure TUrlTestForm.Set_PrintScale(Value: TxPrintScale);
begin
  PrintScale := TPrintScale(Value);
end;

procedure TUrlTestForm.Set_Scaled(Value: WordBool);
begin
  Scaled := Value;
end;

procedure TUrlTestForm.Set_Visible(Value: WordBool);
begin
  Visible := Value;
end;

procedure TUrlTestForm.Set_WindowState(Value: TxWindowState);
begin
  WindowState := TWindowState(Value);
end;

procedure TUrlTestForm.ActivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnActivate;
end;

procedure TUrlTestForm.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TUrlTestForm.CreateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnCreate;
end;

procedure TUrlTestForm.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TUrlTestForm.DeactivateEvent(Sender: TObject);
begin
```

continues

LISTING 25.9 Continued

```
  if FEvents <> nil then FEvents.OnDeactivate;
end;

procedure TUrlTestForm.DestroyEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDestroy;
end;

procedure TUrlTestForm.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TUrlTestForm.PaintEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnPaint;
end;

procedure TUrlTestForm.Label1Click(Sender: TObject);
begin
  HLinkNavigateString(IUnknown(VCLComObject), 'http://www.inprise.com');
end;

procedure TUrlTestForm.Label2Click(Sender: TObject);
begin
  HLinkGoForward(IUnknown(VCLComObject));
end;

procedure TUrlTestForm.Label3Click(Sender: TObject);
begin
  HLinkGoBack(IUnknown(VCLComObject));
end;

procedure TUrlTestForm.Button1Click(Sender: TObject);
begin
  // Note: you may have to change the name of the AVI file shown in the first
  // parameter to Format to another AVI file which resides on your server.
  URLDownloadToFile(IUnknown(VCLComObject),
    PChar(Format('http://%s/delphi3.avi', [ServerName.Text])),
    'c:\temp\testavi.avi', 0, Self);
end;
```

```
initialization
  TActiveFormFactory.Create(ComServer, TActiveFormControl, TUrlTestForm,
    Class_UrlTestForm, 1, '', OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
    tmApartment);
end.
```

The `URLDownloadToFile()` example downloads an AVI file from the server and plays it in a `TMediaPlayer`. Note that this example expects to find a file called `Speedis.avi` in the root of the server (you will find it in the `\Runimage\Delphi50\Demos\Coolstuf` directory of your Delphi 5 CD), so you may need to change the code depending on what AVI files you have on your machine. Figure 25.10 shows this ActiveForm in action inside of Internet Explorer.

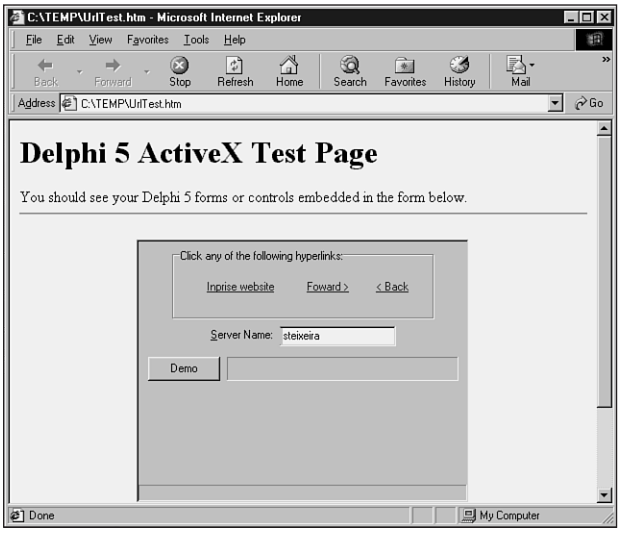

FIGURE 25.10
The ActiveForm running in Internet Explorer.

Web Deployment

The Delphi IDE contains a very convenient feature that helps you deploy your ActiveX projects over the Web. This option is accessible when you are editing an ActiveX project from Project, Web Deployment Options on the main menu. The main page of this dialog is shown in Figure 25.11.

The Project Page

On this page, Target Dir represents the pathname to which you want to deploy the ActiveX project. Note that this assumes you are able to map a drive to your Web server—the contents of

the edit control must be a regular or UNC pathname. Note also that you should not type in a filename, just a path.

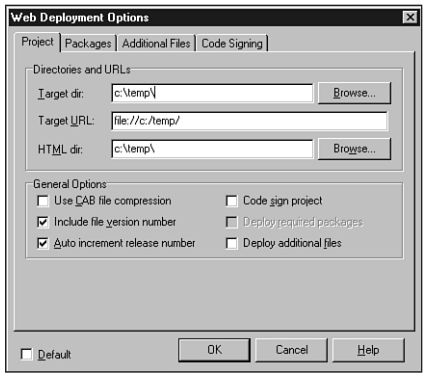

FIGURE 25.11
The Project page of the Web Deployment Options dialog.

Target URL is the URL that references the same directory specified in Target Dir. This must be a valid URL that uses a standard URL prefix (`http://`, `file://`, `ftp://`, and so on). Again, do not include a filename here, just a pathname URL.

HTML Dir is another pathname that dictates where the generated HTML file will be copied. Typically, this is the same as Target Dir.

This dialog also enables you to choose several project deployment options:

- *Use CAB file compression.* Selecting this options will cause your OCX file to be compressed using the Microsoft Cabinet (CAB) format. This is recommended for controls you plan to deploy to clients who use low-bandwidth Web links.
- *Include file version number.* This option indicates whether to include a version number in the generated HTML or INF file. Doing so is recommended, because it provides a means by which users can avoid downloading the control if they already have the most recent version.
- *Autoincrement release number.* When checked, this option causes the release number portion of your `VersionInfo` resources to be automatically incremented after deployment.

NOTE

You must have Internet Explorer 3.02 or greater and Authenticode 2.0 in addition to a certificate from a provider such as VeriSign in order to code-sign files.

- *Deploy required packages.* If your project is built with packages, simply checking this box will automatically include packages used by your project in the file deployment set.
- *Deploy additional files.* By checking this box, you can add files shown on the Additional Files page to your file deployment set.

Packages and Additional Files

The Packages and Additional Files pages are shown in Figures 25.12 and 25.13. The only difference between the pages is that the Packages page is filled automatically based on the packages used by the project, and files are added to and removed from the Additional Files page by you.

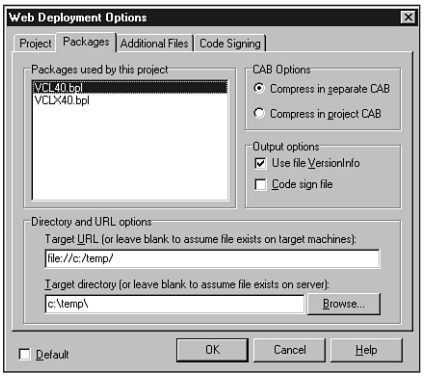

FIGURE 25.12
The Packages page.

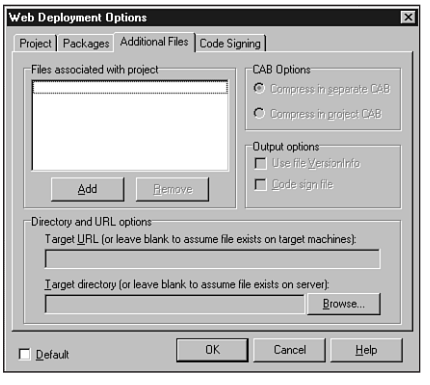

FIGURE 25.13
The Additional Files page.

When you choose to use CAB compression on the Project page, the CAB Options group of the Packages and Additional Files pages enable you to select whether you want the file compressed with the OCX or in a separate CAB file. It is generally more efficient to compress each file in its own CAB, because then the user will not have to download files that they potentially already have installed on their machines. Here are some other options you should be familiar with:

- If the Use File VersionInfo option is selected, the deployment engine will determine whether the selected file has `VersionInfo` and, if so, will stamp the version number contained in `VersionInfo` in the INF file.
- The Target URL edit box will default to the same location as the target URL from the Project page. This is the URL from which the file can be downloaded. If you are assuming that the client of your ActiveX control already has this file installed, leave this value blank.
- The Target Directory edit box allows you to specify the directory to which the particular file should be copied. Leave this blank if the file already exists on the server and should not be recopied to the server.

Code Signing

The Code Signing page, shown in Figure 25.14, allows you to specify the location of the certificate file and private key file associated with your certificate. In addition, you can specify a title for your application, a URL for your application or company, the type of encryption you want to use, and whether to timestamp your certificate. It is recommended that you choose to timestamp as you code-sign so that the signature will remain valid even after your certificate expires.

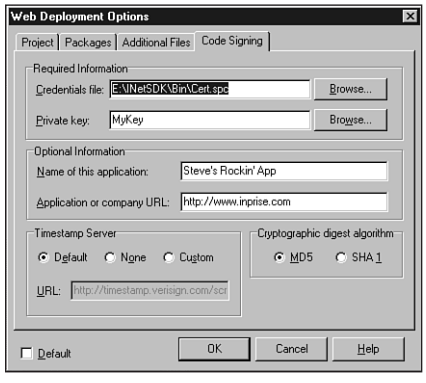

FIGURE 25.14
The Code Signing page.

General Tips

If you make an error on the Project page, your control will usually appear on the Web page as a box with a red × in the upper-left corner. If this happens, you should check the generated HTM file and the INF file (if you are deploying multiple files) for errors. The most common problem is an incorrect URL specified for the control.

Summary

That about sums it up for the topic of creating ActiveX controls and ActiveForms in Delphi. This chapter provided a lot of insight into the inner workings of the ActiveX wizards to help you work within and extend the Delphi ActiveX framework for your benefit. This chapter also built upon the COM and ActiveX knowledge you gained in the previous two chapters—you are well on your way to becoming an expert ActiveX programmer. Now it's time to change gears. The next chapter, "Using Delphi's Open Tools API," focuses on using Delphi's Open Tools API to get inside the IDE.