ActiveX

Supporting distributed computing

Components and Object-Oriented Analysis and Design (OOAD)

- Component - reusable binary code
- Breaking an application into components, each of which can be modified and updated without affecting other components
e.g. PowerPoint: text area, tool bar, menu bar, spell checker, thesaurus, equation editor, graphic editor, etc.

COM

- COM - binary-level specification that describes how a series of components can communicate
- Components can be written in different programming languages by different vendors
- Interoperability - the ability of disparate components to intercommunicate

COM (continued)

- Application interacts with components through a set of standard methods grouped into interfaces
- Different types of interfaces exist for ActiveX controls, transferring data, storing information, etc.
- Most fundamental interface is IUnknown
- Every COM interface must contain IUnknown methods: AddRef, Release, QueryInterface

COM (continued)

- IUnknown methods
  - AddRef and Release - control a component’s lifetime
  - QueryInterface - real-time query if a component supports a particular COM interface (e.g., one component needs to know if another supports drag-and-drop operation)
- VB programmers do not deal with IUnknown methods
- COM+ - an enhanced version of COM

DCOM - COM “with a long wire”

- Distributed COM that provides the means of executing components on remote machines
- An extension of COM - communication accomplished through the same COM mechanism
- Components created with VB can run remotely using DCOM
Object Linking and Embedding

- OLE is a technology that allows applications to work together seamlessly
- Provides a number of services: drag and drop, automation (one application controls another)
- Some OLE controls allow in-place activation (or visual editing) - the OLE acts as a host container for another control

ActiveX

- Built on OLE, COM, and DCOM
- Improves OLE
  - providing distributed capabilities
  - providing more efficient design

ActiveX controls
- ActiveX documents
- ActiveX components

ActiveX controls (continued)

- Three types of ActiveX controls can be created in VB
  - user drawn controls - created from scratch
  - "enhanced" controls - adding capability to existing controls (e.g., MaskEdit an enhanced version of Text Box)
  - aggregate controls - contains one or more intrinsic, ActiveX, third party) controls
- aggregate controls most common

ActiveX controls

- reusable components that can be plugged into a VB program
- built on OLE and VBX controls (VBX is an older technology replaced by ActiveX)
- examples of ActiveX controls: calendar, word processor, image editor, card game
- can be visible or invisible

Light weight controls

- Controls that do not have a window handle (an identification automatically created by Windows OS)
- A control with a window handle (stored in control’s hWnd property) is called a windowed controls
- Lightweight controls are created at design time with the Windowless property set to True (referred to as windowless controls)
Light weight controls (continued)

- hWnd property is a run-time property
- light weight controls can only contain other light weight controls
- light weight controls consume fewer resources, but not necessarily execute any faster
- light weight controls are created for distributed applications
- Not universally supported. If not supported, will run in wondowed mode. IE 4.0+ supports light weight controls

ActiveX control Lifetime and Events

- An instance created when double clicking or drawing a control on a form
- This involves
  - instantiation of each constituent controls
  - The UserControl object is created. Constituent controls are sited on UserControl
  - Event UserControl_Initialize is executed
  - The first time ActiveX control sited in container, and UserControl_InitProperties executed

ActiveX control Lifetime and Events (continued)

- If an instance of an ActiveX control already exists (e.g. in an existing form), UserControl_ReadProperties is executed
- Event UserControl_Resize is called to resize the control
- Show and Paint events are raised. Control becomes visible
- Programmer writes code for control’s Public events
- Program runs (design phase ends) and form is closed, design properties saved in .frm file

ActiveX control Lifetime and Events (continued)

- Control is unsited (removed from container)
- UserControl_Terminate is called.
- Initialize is called. Any constituent controls are created but not sited, yet.
- ReadProperties is called to retrieve saved property values.
- control is sited on the run-time instance of the container.
- Resize or Paint events are raised. The control sizes any constituent controls. User-drawn controls draw themselves.

ActiveX control Lifetime and Events (continued)

- During program execution, developer can change control properties.
- When program is terminated, run-time instances of the container and control are destroyed. Terminate is called before control instance is destroyed.
- Run-time setting of property values is not saved.
- The container and control design-time instanced are created again.
- Closing design-time container or closing the project results in calling WriteProperties and Terminate.

UserControl Object

- Creating an ActiveX control involves loading a UserControl object into an ActiveX control project
- UserControl is a container where other controls can be sited (like a form)
- UserControl behaves programmatically different from a form - form events like Activate, DeActivate, Load, UnLoad, QueryUnload do not exist. Only Terminate is similar to UnLoad.
- UserControl files has .ctl extension.
ActiveX Document Creation

- When you create a new ActiveX document, the steps you'll generally follow are:
  - Determine the features your document will provide.
  - Design the appearance of your document and the interface for your document; that is, the properties, methods, and events your document will expose.
  - Create a project consisting of your user document and any auxiliary forms.
    - Add controls and/or code to the UserDocument object.
    - Implement the interface and features of your document.
    - Compile your document to create a .vbd file and test it with all potential target applications.

Change event

- Indicates the contents of a control have changed. How and when this event occurs varies with the control:
  - HScrollBar and VScrollBar — move the scroll box portion of the scroll bar. Occurs when the user scrolls or when you change the Value property setting through code.
  - Label — changes the contents of the Label. Occurs when a DDE link updates data or when you change the Caption property setting through code.
  - TextBox — changes the contents of the text box. Occurs when a DDE link updates data, when a user changes the text, or when you change the Text property setting through code.

UserControl Change event

Public Event Change ()

Private Sub UserControl_Resize()
  lblDisplay.Caption = hsbScroll.Value
  RaiseEvent Change
End Sub

- User determines whether to respond to the raised change event by defining or not defending the subroutine change

AmbientProperties Object

To stay consistent with the container

- Containers provide ambient properties to give controls hints about how they can best display themselves in the container.
  - E.g., the ambient BackColor property tells a control what color to set its own BackColor property to in order to blend in with the container.
- VB makes ambient properties available to your ActiveX control through an AmbientProperties object. The Ambient property of the UserControl object returns a reference to the AmbientProperties object.

AmbientProperties Object

- The AmbientProperties object provided by VB contains all of the standard ambient properties defined by the ActiveX Controls Standard, whether or not they are actually provided by the container your control instance was placed on.
- You can safely access any of the properties of the AmbientProperties object visible in the Object Browser. If an ambient property is not provided by the container, the AmbientProperties object returns a default value, as listed in the topics for the AmbientProperties object properties.

Compile your ActiveX control

- To make the control you created a stand-alone control that can be run outside the IDE
- Remove any forms used to test the control
- Open the Project Property dialog, and set
  - Project Type = ActiveX
  - Startup Object = None
  - Project Name = ctlClock
  - Project Description = VB6HTP1 Clock ActiveX Control (this appears as the name of the control in the Components dialog).
- Close the Project Property dialog
Setting project properties

 Compile your ActiveX control (cont.)

- From File menu, select Make VB6HTP1 Clock.ocx
- Select a location to save the project
- VB automatically register the control

ActiveX DLLs

- ActiveX objects that are not executed by itself but run by other applications
- ActiveX Dynamic Link Library
- Create ActiveX DLL by opening an ActiveX DLL project
- Write code
- Save as a DLL by selecting Make <project name>.dll from File menu
- Use by selecting it from the Project menu Reference

ActiveX DLL - spell checker

- Figure 17.47
- ActiveX DLL example
- Option Explicit
- Private mWordRef As Word Application
- Private Sub Class_Initialize()
  Set mWordRef = New Word.Application
- End Sub
- Function returns False if s is spelled incorrectly
  Public Function SpellCheckWord(ByVal s As String) As Boolean
    SpellCheckWord = mWordRef.CheckSpelling(s)
- End Function
- Private Sub Class_Terminate()
  Call mWordRef.Quit
  Set mWordRef = Nothing
- End Sub

Use of ActiveX DLL - spell checker

- Form module to exercise SpellChecker DLL
  Option Explicit
  Private mChecker As CWordChecker
  Public Sub Form_Initialize()
    Set mChecker = New CWordChecker
- End Sub
  Private Sub cmdCheck_Click()
    If mChecker.SpellCheckWord(txtInput.Text) Then
      lblDisplay.Caption = txtInput.Text & " is spelled correctly."
    Else
      lblDisplay.Caption = txtInput.Text & " is spelled incorrectly."
- End If
- End Sub
  Private Sub Form_Terminate()
    Set mChecker = Nothing
- End Sub

ActiveX EXEs

- A VB project type for creating applications that can be either run as stand-alone applications or be executed through automation (e.g. Word)
- ActiveX DLLs run in-process (they are loaded together with the application using them into the memory)
- ActiveX EXEs do not run in-process. They use marshaling (interprocess communication) to access resources of other processes.
Creating ActiveX EXE

- Open a new ActiveX EXE project
- A class module is automatically created
- Code the class
- Add modules and/or GUI if any (if not automated)
- Must add a standard module with a `Main` sub

Creating ActiveX EXE (cont.)

- For stand-alone ActiveX EXE, use `Show` to load form
  
  ```vb
  Procedure Sub Main()
  If App.StartMode = vbSModeStandAlone Then
  Call frmCheck.Show
  Else
  Set mChecker = New CSpellCheckerEXE
  End If
  End Sub
  ```

- `App` object is used to get information about current application

Creating stand-alone ActiveX EXE

- Save as an ActiveX EXE by selecting `Make <project name>.exe` from *File* menu
- Use by double-clicking on the program icon in Windows Explorer

ActiveX EXE run through automation

*Friend* access specified

- Besides `Private` and `Public`, VB allows a method of a class to be accessed by all other modules in the same project through *Friend* access
Program Excel in VB

» Select from menu Project/References
  Microsoft Excel 8.0 (for Office 97) or 9.0 (for Office 2000)

» Open Object Browser (from View). All objects that begin *xl* are Excel objects

» Find help by conducting a search using *Excel* as keyword

» Consider to use MSChart for graphing (reference library Microsoft Graph 8.0)
  (search for MSChart in on-line help)

Program Excel in VB - example

Private Sub Command1_Click()
  ' Declare object variables for Microsoft Excel,
  ' application workbook, and worksheet objects.
  Dim xlApp As Excel.Application
  Dim xlBook As Excel.Workbook
  Dim xlSheet As Excel.Worksheet

  ' Assign object references to the variables. Use
  ' Add methods to create new workbook and worksheet
  ' objects.
  Set xlApp = New Excel.Application
  Set xlBook = xlApp.Workbooks.Add
  Set xlSheet = xlBook.Worksheets.Add

  ' Assign the values entered in the text boxes to
  ' Microsoft Excel cells.
  xlSheet.Cells(1, 1).Value = Text1.Text
  xlSheet.Cells(2, 1).Value = Text2.Text

  ' Use the Formula method to add the values in
  ' Microsoft Excel.
  xlSheet.Cells(3, 1).Formula = "=R1C1 + R2C1"
  Text3.Text = xlSheet.Cells(3, 1)

  ' Save the Worksheet.
  xlSheet.SaveAs "c:\Temp.xls"

  ' Close the Workbook
  xlBook.Close

  ' Close Microsoft Excel with the Quit method.
  xlApp.Quit

  ' Release the objects.
  Set xlApp = Nothing
  Set xlBook = Nothing
  Set xlSheet = Nothing
End Sub

Excel Chart (to be continued)

Dim xlChart As Excel.Chart