Lab 5: Using Dynamic-Link Libraries

For background information on this lab, click each of these topics:

Objectives

In this lab, you will call two Windows DLLs to find the location of the Windows folder and create a topmost window. In addition, you will call a Windows timer by using a callback function.

After completing this lab, you will be able to:

- Declare a DLL.
- Call a function of a DLL.
- Manipulate strings returned from a DLL.
- Locate the Windows folder by using the Windows API.
- Create a topmost window by using the Windows API.
- Use Windows timers without a form on which to place a timer control.

Prerequisites

Before working on this lab, you should be familiar with the following concepts:

- The API Text Viewer
- Standard modules
- The contents of this chapter

Lab Setup

To complete this lab, you need the following setup:

- Visual Basic version 5.0 or later

To see a demonstration of the completed lab solution, click this icon.

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 5.

Exercise 1: Locating the Windows Folder

In this exercise, you will use the Windows API function **GetWindowsDirectory** to find the folder that contains Microsoft Windows.

Exercise 2: Creating a Topmost Window

In this exercise, you will use the Windows API function **SetWindowPos** to create a topmost window (or form) that remains on top of other windows, regardless of the current focus.

Exercise 3: Using Callbacks

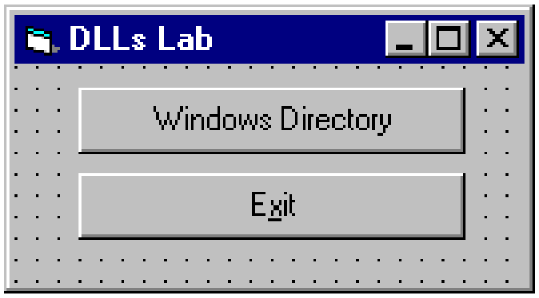
In this exercise, you will use the **AddressOf** operator to establish a callback function to receive timer notifications.

Exercise 1: Locating the Windows Folder

In this exercise, you will use the Windows API function **GetWindowsDirectory** to find the folder that contains Microsoft Windows.

▶ Create a form

1. Create a new Standard EXE project.
2. Save the project in the <*Install Folder*>\Labs\Lab05.
3. Add controls to the form, as shown below.



▶ Declare a procedure

1. Add a standard module to the project.
2. From the API Viewer, copy the **GetWindowsDirectory Declare** statement.

   For more information about copying declarations, see Using the API Viewer.
3. Paste the **Declare** statement into the new module.

▶ Create an event procedure

1. Create a Click event procedure for the Windows Directory button.
2. In the event procedure, add code that will display the path to the Windows folder in a message box.

   Be sure to follow the rules for using strings, as discussed in Passing Strings.

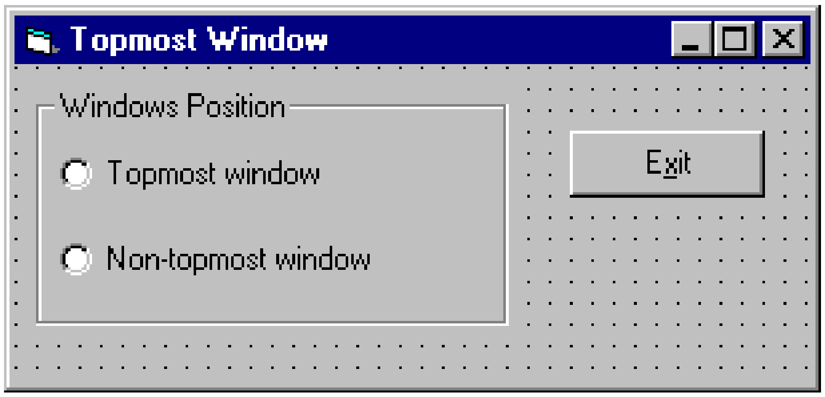
```
'DLL Declarations
Declare Function GetWindowsDirectory Lib "kernel32" _
    Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, _
    ByVal nSize As Long) As Long
Private Sub cmdWinDir_Click()
    Dim WinDir As String
    Dim ReturnSize As Long
    WinDir = String(255, 0) 'fill string w/nulls
    'call API to get win directory and capture the
    'size of the return string
    ReturnSize = GetWindowsDirectory(lpBuffer:=WinDir, _
        nSize:=Len(WinDir))
    'trim the string down to the correct size
    WinDir = Left(WinDir, ReturnSize)
    MsgBox "The Windows directory is: " & WinDir
End Sub
```

3. Save and test the project.

Exercise 2: Creating a Topmost Window

In this exercise, you will use the Windows API function **SetWindowPos** to create a window or form that remains on top of other windows, regardless of the current focus.

1. Create a new project.
2. Save the project in the <*Install Folder*>\Labs\Lab05.
3. Add controls to the form, as shown below:



4. Create event procedures for the **Topmost window** and **Non-topmost window** option buttons.

```
Private Sub optNonTopmost_Click()
  SetWindowPos frmtopmost.hwnd, _
    HWND_NOTOPMOST, 0, 0, 0, 0, FLAGS
End Sub
Private Sub optTopmost_Click()
  SetWindowPos frmtopmost.hwnd, _
    HWND_TOPMOST, 0, 0, 0, 0, FLAGS
End Sub
```

5. Save and test the application.

Exercise 3: Using Callbacks

In this exercise, you will use the **AddressOf** operator to establish a callback function to receive timer notifications.

▶ **Create the initial project**

1. Create a new .exe project and save it in the <*Install Directory*>\Labs\Lab05.
2. Add the component for the Microsoft Windows Common Controls.
3. Place a **ProgressBar** control on the form.
4. Place a command button on the form.
5. Add a module to the project and save the project.

▶ **Add code to start the timer**

1. Add declarations to the module for the **SetTimer** and **KillTimer** Windows API functions.

   You can use the API Viewer that ships with Visual Basic to make this task easier.
2. Create a **Sub** procedure named **StartTimer** and add it to the form.
3. Declare a form-level integer variable to hold the return value of the timer ID.
4. Call the **SetTimer** function with the parameter settings as shown in the following table.

Parameter	Value	Comment

HWnd	0	Not used when specifying a callback routine.
idEvent	0	Not used when specifying a callback routine.
Interval	200	The interval, in milliseconds, at which the timer while fire.
Callback	Addressof TimerProc	The address of the timer callback procedure; the procedure with the name TimerProc.

5. Set the value of the **ProgressBar** to zero.
6. Set the caption on the command button to **Stop**.

▶ **Add code to end and reset the timer**

1. Create a **Sub** procedure named **EndTimer** and add it to the form.
2. If the timer ID is not 0, call the **KillTimer** function with the appropriate parameter values.
3. Set the timer ID variable to 0.
4. Set the caption on the command button to **Start**.
5. Add a Form_Unload event handler that calls **EndTimer**.

▶ **Code the controls on the form**

1. In the Click event of the Command button, test for the following conditions:
   a. If the timer ID variable is 0, call the **StartTimer** procedure.
   b. If the timer ID variable is not 0, call the **EndTimer** procedure.
2. Create a public **Sub** procedure named **UpdateProgressBar**.
3. Calculate the next **ProgressBar** value.
   This is the current value plus some increment; an increment of five works well here.
4. Test for the following conditions:
   a. If the next value is 100 or greater, set the **ProgressBar** value to 100 and call **EndTimer**.
   b. If the next value is less then 100, set the **ProgressBar** value to the calculated value.

```
Declare Function SetTimer Lib "user32" (ByVal hwnd As Long, ByVal _
    nIDEvent As Long, ByVal uElapse As Long, _
    ByVal lpTimerFunc As Long) As Long
Declare Function KillTimer Lib "user32" (ByVal hwnd As Long, ByVal _
    nIDEvent As Long) As Long
Private idTimer As Integer
Const cIncrement As Integer = 5
Private Sub cmdTimer_Click()
  ' The timer command button simply toggles between starting
  ' and ending the timer.
  If idTimer > 0 Then
    EndTimer
  Else
    StartTimer
  End If
End Sub
Public Sub UpdateProgressBar()
  ' This public function is called by the timer callback routine
  ' to increment the precentdone bar. If the value is equal to
  ' or exceeds 100, we terminate the timer.
  Dim PercentDone As Integer

  PercentDone = ProgressBar1.Value + cIncrement
  If PercentDone >= 100 Then
    ProgressBar1.Value = 100
    EndTimer
  Else
    ProgressBar1.Value = PercentDone
  End If
End Sub
Private Sub StartTimer()
  ' Start the timer. Note the use of the AddressOf operator
  ' to pass the address of the timer procedure to Windows. When
  ' the timer fires, the callback procedure will be called. The
  ' timer will continue to fire at 200 millisecond intervals
  ' until it is explicitly terminated with a call to KillTimer.
  ' Because we are using a callback, the first two parameters of
  ' SetTimer, the HWnd and event ID, are not used.
  idTimer = SetTimer(0, 0, 200, AddressOf TimerProc)
  ProgressBar1.Value = 0
  cmdTimer.Caption = "&Stop"
End Sub
Private Sub EndTimer()
  ' Kill the timer and reset to start again
  KillTimer 0, idTimer
  idTimer = 0
  cmdTimer.Caption = "&Start"
End Sub
```

▶ Add the timer callback function

1. In the module, add the callback function with the following syntax:

   ```
   Public Sub TimerProc(ByVal hwnd As Long, ByVal msg As Long, _
     ByVal idEvent As Long, ByVal curTime As Long)
   ```

2. In the callback function, call the form's **UpdateProgressBar** function.
3. Save and test the application.

   The **Start** button activates the timer and invokes the callback procedure. The timer continues to fire at intervals of 200 milliseconds (you can see this as the progress bar values update) until the callback is explicitly terminated by a call to the **KillTimer** function.