Create an Application Launcher

Make sure your user runs the latest version of your app after you make changes to the code.

by Chris Barlow

WHAT YOU NEED
Visual Basic 5.0 or 6.0

Click & Retrieve Source CODE!

You've completed your Visual Basic system and installed it on the user's computer. It consists of a database and three different applications. You've put shortcuts to all three applications on the user's desktop. The user has double-clicked on the shortcuts and seen the database open and each application start up. The user is satisfied, so you're all done, right?

If you've ever been through a system startup, you know the answer. You're not even close to "all done"—in some ways, this is just the beginning. I guarantee the user will soon call and tell you one of the applications isn't working, or he wants a "minor" change in one of the screens. Or you'll find a little code you want to tweak or a feature you want to improve.

Welcome to the world of system maintenance. Fortunately, Visual Basic makes it easy to implement the changes users invariably want as they use the system day to day. You can change menus, redraw screens, and even add new fields to the database, almost faster than users can tell you what they want. Most of these changes require only a simple compile of a new EXE—not an entirely new setup kit.

Although making the changes is easy, you need to get the new, updated version of your application in place before the user can launch it. This isn't as easy as it sounds. Often you don't visit the user's site to install the new version; instead, you connect to his or her computer through a modem, LAN, or the Internet. You might connect directly to the user's disk to overwrite your old EXE, or you might simply see a shared server on the user's network. You can't have your applications reside on the server and point each user's desktop shortcuts to the server—suppose the user is running the application when you want to replace it. You'll get an error message telling you the file is open and cannot be overwritten.

You can solve this problem by creating an application launcher. Whenever a user wants to run one of the applications in the system, his or her request is funneled through the launcher. Unlike a typical, rigid menu application, in most cases the user doesn't even realize the application launcher is being executed. Its job is to make sure the latest version of the app is running.

Because the desktop shortcuts point to the launcher rather than directly to the application, the launcher fires up when the user double-clicks on the shortcut to launch an application. The launcher looks at the version information from the application's EXE file in the program folder. Then the launcher looks for the same EXE file in a predefined install folder on that computer or a server and checks the version information of that file. If the EXE file in the install folder is newer, the launcher copies the newer version into the program folder. Finally, the launcher gives the command to launch the application, and terminates.

The launcher makes program upgrades easy. Simply copy a new version of the app to the install folder after it is compiled. The launcher copies your new version to the program folder and launches it the next time the user runs the app.

You can also centralize functionality in the launcher because requests to run your applications are funneled through it. For example, you could use the launcher to log the user onto your system by requesting a user ID and password, and verifying this information against a database. You could even "lock up" your applications so they won't start unless they receive a certain code on the command line from the launcher. That way, the user could not run the application directly, but would always have to go through the launcher.

Create a Launcher

You must control the version of your application to use the launcher. Visual Basic makes it easy to identify your program by adding a version resource to the EXE file.

Use the Options button on the Make Project window to modify the information about your app. I suggest setting the option to automatically increment the revision each time you compile the application. You can override the Minor and Major properties as you make substantial changes to your application.

In my August 1997 *VBPJ* column, "Identify an App's Version," I wrote an ActiveX DLL that reads this version information, along with the file's date and time. The launcher uses this FileVer DLL to read the version resource information from the file in both the program and install folders to determine which is more recent. You can download this DLL from The Development Exchange (see the Download Free Code box at the end of this column for details).

To create the launcher, start a new Standard EXE project in Visual Basic, and right-click on the Project window to add a module. Click on the Properties menu item on the Project menu, and name the project Launcher. Set the Startup Object to Sub Main. Then click on the References menu item on the Project menu to add a reference to the FileVer DLL.

VB's Standard EXE project has a single form you can use as a splash screen to let the user know what's happening while you copy the upgraded EXE file. Rename it Splash.frm. I suggest designing a simple form with Label controls for the launcher's version and copyright, as well as a larger Label control to display the action that the launcher is performing.

The Main subroutine executes when the launcher starts. First, show the Splash form so the user has some visible feedback. Use the GetSetting statement to get the setting for the install folder from the registry, and store the setting in a public variable called InstallFolder. If this variable is null, as it will be the first time the launcher runs on a computer, use the InputBox function to request the install folder. If the user doesn't enter a folder or clicks on the Cancel button, the launcher will terminate; otherwise, the install folder setting is saved to the registry:

```
Public Sub Main()
Dim txt$
Splash.Show
Splash.Refresh
InstallFolder = GetSetting("Launcher", _
   "Folders", "InstallFolder", "")
If Len(InstallFolder) = 0 Then
   InstallFolder = InputBox _
      ("Please enter path to " & _
      "Install Folder", _
      "No Install Folder", App.Path)
   If Len(InstallFolder) = 0 Then End
   SaveSetting "Launcher", "Folders", _
      "InstallFolder", InstallFolder
End If
```

The application to be launched is passed on the command line with the full path to the application. Set the Command Line Arguments on the Make tab of the Project Properties dialog to test this in debug mode (see Figure 1). VB makes command line arguments available through the Command property of the App object. If this property is null, then the user hasn't specified a program to launch, and an appropriate error message is displayed:

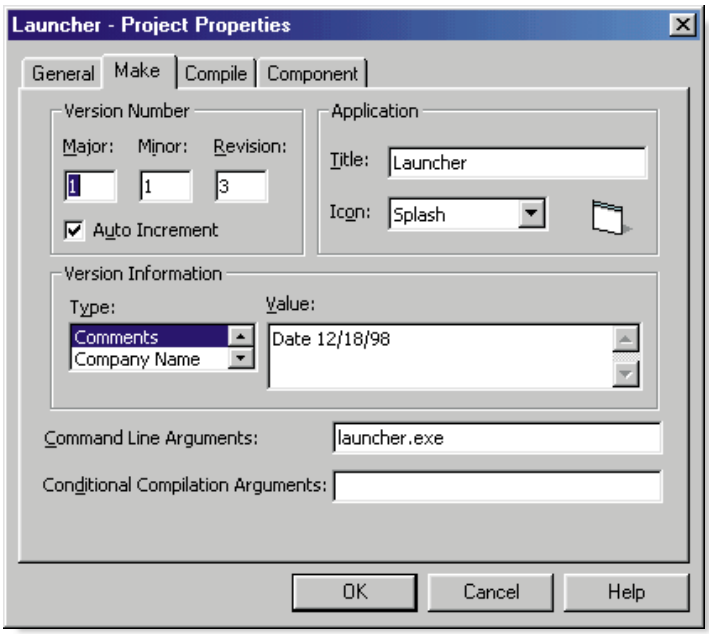

Figure 1 Your Project's Make Properties. This dialog lets you set the Command Line Arguments that you need to test the launcher application in the Visual Basic design environment.

```
If Len(Command) = 0 Then
   txt = "This program will " & _
      "launch the program " & _
      "specified on the command " & _
      "line." & vbCrLf
   txt = txt & "It will compare " & _
      "this program's date and " & _
      "version against the same " & _
      "program in " & vbCrLf
   txt = txt & InstallFolder & vbCrLf
   txt = txt & "and give you the " & _
      "option to replace your " & _
      "program with this version." & _
      vbCrLf
   txt = txt & "Please enter path " & _
      "and EXE name on command " & _
      "line and re-run this program."
   MsgBox txt
```

The Command property is parsed at the first space if there is any text on the command line, in case additional command line arguments need to be passed to the launched program (see Listing 1 for the

VB5 or VB6 | Launcher Module

```
Option Explicit
Public Const Prognme = "Launcher" 'Program Name
Public Const ProgVer = "1.01.05"  'Program Version

Public InstallFolder As String

Public Sub Main()
Dim txt$
Splash.Show
Splash.Refresh
InstallFolder = GetSetting("Launcher", "Folders", _
  InstallFolder", "")
If Len(InstallFolder) = 0 Then
  InstallFolder = InputBox( _
    "Please enter path to install folder", _
    "No Install Folder", App.Path)
  If Len(InstallFolder) = 0 Then End
  SaveSetting "Launcher", "Folders", "InstallFolder", _
    InstallFolder
End If
If Len(Command) = 0 Then
  txt = "This program will launch the program " & _
    "specified on the command line." & vbCrLf
  txt = txt & "It will compare this program's date " & _
    "and version against the same program in " & _
    vbCrLf
  txt = txt & InstallFolder & vbCrLf
  txt = txt & "and give you the option to replace " & _
    "your program with this version." & vbCrLf
  txt = txt & "Please enter path and EXE name on " & _
    "command line and re-run this program."
  MsgBox txt
Else
  If ProgCheck(ParseString(Command, " ", 1)) Then
    Shell Command, 1
  End If
  Unload Splash
End If
End
End Sub

Private Function ProgCheck(File$) As Boolean
Dim verLocalF$, res%
Dim verServerF$
Dim prgServerF$, msg$
res = FileExist(File)
If res = True Then
  verLocalF = GetProgVersion(File)
Else
  msg = File & " is not in Program folder." & vbCrLf
End If
prgServerF = InstallFolder & "\" & _
  StripPathName(File)
res = FileExist(prgServerF)
If res = True Then
  verServerF = GetProgVersion(prgServerF)
  On Error GoTo ErrorHdler
  If verLocalF < verServerF Then
    Splash.lbComment = "Copying from Server"
    Splash.Refresh
    FileCopy prgServerF, File
    msg = ""
  End If
  ProgCheck = True
ElseIf res = 53 Then
  If Len(msg) = 0 Then ProgCheck = True
  msg = msg & prgServerF & _
    " is not in Install folder."
End If
If Len(msg) Then MsgBox msg
Exit Function

ErrorHdler:
If Err = 13 Or Err = 70 Then
  msg = "Error! This program cannot be copied." & _
    vbCrLf
  msg = msg & "Possible Reason:" & vbCrLf
  msg = msg & "1. Your application may be " & _
    "currently running" & vbCrLf
  msg = msg & "2. Permission denied, " & _
    "no file copied" & vbCrLf
  msg = msg & "Solution:" & vbCrLf
  msg = msg & "1. Exit the current running program " & _
    "and run the program again" & vbCrLf
  msg = msg & "2. Continue to use the existing " & _
    "program, but you won't get the new version"
  MsgBox msg, vbExclamation, "Copy Error"
End If
Exit Function
End Function

Private Function GetProgVersion(File$) As String
Dim cF As New cFileVer

On Error GoTo Errhdlr
cF.sFileName = File$
cF.GetFileVersionData
GetProgVersion = cF.sFileVersion

Exit Function

Errhdlr:
  MsgBox "Error= " & Error$ & " Err=" & _
    Err.Number & Chr(13) & "File name=" & _
    "(" & File$ & ")"
End Function

Function ParseString(s$, del$, n) As String
  Dim pos As Long, i As Integer, pos2 As Long
  ParseString = s$
  pos = InStr(s$, del$)
  If pos Then'if has del$
    If n = 1 Then
      ParseString = Left$(s$, pos - 1)
    Else
      For i = 1 To n - 1  'count items
        pos2 = InStr(pos + 1, s$, del$)
        If pos2 = 0 Then  'end of string
          If i = n - 1 Then
            ParseString = Trim(Mid$(s$, pos + _
              Len(del$)))   'len of delimiter
          Else
            ParseString = ""  'nth item not found
          End If
          Exit Function
        End If
        ParseString = Trim(Mid$(s$, pos + _
```

Continued on page 69.

Listing 1 This module contains the ParseString procedure, which you use to parse the Command property.

ParseString procedure). The file path is passed as an argument to a ProgCheck procedure to check the version information. The application launches using the Shell statement if the ProgCheck procedure returns True. Finally, the Splash form is unloaded, and the launcher terminates:

```
Else
   If ProgCheck(ParseString(Command, _
      " ", 1)) Then Shell Command, 1
   End If
   Unload Splash
End If
```

Continued from page 68.

```
            Len(del$), pos2 - pos _
            - Len(del$)))
          pos = pos2
       Next
    End If
  ElseIf n > 1 Then
    ParseString = ""
    'nth item not found
  End If
End Function

Function FileExist(fname) As _
  Integer
'checks if file exists,
'returns True or error
  Dim FileRet As Integer
  On Error Resume Next
  'first check for directory
  FileRet = Len(Dir(fname, _
    vbNormal))
  'SJS 1/13/97 -
  'add vbNormal parameter
  If Err Then
    FileExist = Err
  ElseIf FileRet Then
    FileExist = True
  Else
    FileExist = 53
  End If
  On Error GoTo 0
End Function

Function StripPathName(fname _
  As String) As String
'returns filename
On Error Resume Next
Dim i As Integer
For i = Len(fname) To 1 Step -1
  If Mid(fname, i, 1) = "\" Then
    Exit For
  End If
Next
StripPathName = Trim(Mid(fname, _
  i + 1))
End Function
```

```
End
End Sub
```

Checking the Version

The ProgCheck procedure needs to check if the file exists in both the program and install folders. The ProgCheck procedure also checks the versions and copies the EXE from the install folder if it is a newer version. First, check for the file in the program folder and get its version with the GetProgVersion procedure:

```
Private Function ProgCheck(File$) _
   As Boolean
Dim verLocalF$, res%
Dim verServerF$
Dim prgServerF$, msg$
res = FileExist(File)
If res = True Then
   verLocalF = GetProgVersion(File)
Else
   msg = File & " is not in " & _
      "Program folder." & vbCrLf
End If
```

Next, check for the file in the install folder, get its version, and compare the two versions. If the program folder version is lower, put a message in the Comment label on the Splash form and use the FileCopy statement to copy the file:

```
prgServerF = InstallFolder & "\" & _
   StripPathName(File)
res = FileExist(prgServerF)
If res = True Then
   verServerF = _
      GetProgVersion(prgServerF)
   On Error GoTo ErrorHdler
   If verLocalF < verServerF Then
      Splash.lbComment = _
         "Copying from Server"
      Splash.Refresh
      FileCopy prgServerF, File
      msg = ""
   End If
   ProgCheck = True
```

The procedure ends with appropriate error handling.

The GetProgVersion procedure is simple because it uses the FileVer DLL. Simply dimension an instance of the cFileVer class and set the sFileName property. Then call the GetFileVersionData method, which returns the sFileVersion property:

```
Private Function _
   GetProgVersion(File$) As String
Dim cF As New cFileVer

On Error GoTo Errhdlr
cF.sFileName = File
cF.GetFileVersionData
GetProgVersion = cF.sFileVersion

Exit Function

Errhdlr:
   MsgBox "Error= " & Error$ & _
      " Err=" & Err.Number & _
      Chr(13) & "File name=" & _
      "(" & File$ & ")"
End Function
```

You can download the files for the launcher from The Development Exchange (see the Download Free Code box for details). I'm curious to see how you enhance the launcher. How about looking at a Web server rather than an install folder to check for the latest app? Then you could update your apps over the Internet! VBPJ

Editor's Note: *This column has been updated from Chris' September 1997 Getting Started column. The code is updated for VB6 and includes a few bug fixes.*

About the Author

Chris Barlow, a recognized expert in the Internet, Web, messaging, and applications development, is a frequent speaker at VBITS, Tech•Ed, and DevDays. Chris holds degrees from Harvard Business School and Dartmouth College. Reach him at Chris@VBExpert.com.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from www.vbpj.com.

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the VBPJ home page, and type **VBPJ0499** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the code files for the launcher app and the FileVer DLL from the August 1997 Getting Started column.

To get the code for this article only, available to DevX Premier Club members, type **VBPJ0499GS** into the Locator+ field.