Improving the Shell Function

You can use the Shell function to run another executable program from within a Visual Basic program. This tip shows you a simple way to improve on Shell's functionality.

Shell takes two arguments. The first is the name, including path, of the EXE file to run. The second argument specifies the window style of the program. This argument is optional - if omitted, the program is started minimized with the focus. You can refer to the Visual Basic help for more information on the window style argument because it is not relevant here. The techniques presented here work regardless of the program's window style and even if the program does not have a window.

Shell runs its target program asynchronously, meaning that execution can return to the Visual Basic program before the second program is finished executing. In many situations this is not a concern, but at other times it may be. One example is when your Visual Basic program is dependent in some way on the result of the shelled program completing its operation. In this case you need some way to pause the Visual Basic program until the other program is done.

The first step is to get the handle of the shelled program. The Shell function returns its program ID, or PID, and once you have that you can get the handle using the OpenProcess API function:

```
Public Declare Function OpenProcess Lib "kernel32" _
 (ByVal dwDesiredAccess As Long, _
 ByVal bInheritHandle As Long, _
 ByVal dwProcessId As Long) As Long
```

Pass the value &H100000 (usually represented by the constant SYNCHRONIZE) for the first argument and 0 (as a long) for the second. The third argument is the PID returned by the Shell function. The function's return value is a handle to the Windows process representing the shelled program. With this handle, you can use the WaitForSingleObject API function to pause the Visual Basic program until the shelled program has terminated. The declaration is:

```
Private Declare Function WaitForSingleObject Lib _
"kernel32" (ByVal hHandle As Long, _
ByVal dwMilliseconds As Long) As Long
```

The first argument is the handle of the program to wait for, obtained from the OpenProcess function. The second argument is the length of time to wait. If you pass a millisecond value, the function returns when the shelled program completes or when the specified interval is up, whichever occurs first. If you want the function to wait as long as necessary for the shelled program to complete, pass the value &HFFFF (often represented by the constant INFINITE).

Here is sample code that shows how to use these API functions to shell a program and wait for it to complete.

Const SYNCHRONIZE = &H100000
Const INFINITE = &HFFFF

```
Private Sub ShellProgramAndWait(ProgramName As String)

        Dim hHandle As Long, pid As Long

        txtStatus.Text = "Processing"
        txtStatus.Refresh()
        pid = Shell(ProgramName, vbNormalFocus)
        If pid <> 0 Then
            hHandle = OpenProcess(SYNCHRONIZE, 0&, pid)
            WaitForSingleObject(hHandle, INFINITE)
            txtStatus.Text = "Finished"
        Else
            txtStatus.Text = "Error shelling " & ProgramName
        End If

    End Sub
```

The WaitForSingleObject function effectively freezes your Visual Basic program, something you need to take into account so the user does not think something is wrong. In this code, for example, a message is displayed to the user in a Text Box control. Note the use of the Refresh method to make sure the message is actually displayed before the program enters its wait state. Then, when the shelled program terminates, a "finished" message is displayed. With the techniques presented here, Visual Basic's Shell function becomes an even more useful tool.