Interactive Shelling Between Visual Basic And Blaise

John P. Cashwell, Battelle

The Blaise survey processing system is an effective solution for collecting survey data, however complicated studies may require more sophisticated project management and tracking functionality. The required functionality may be offered by a Visual Basic application. This application could operate as a shell around a Blaise computer-assisted instrument (CAI), helping users to comply with project timelines and parameters.

One of the challenges in developing such a system is creating a streamlined method for shelling from Visual Basic to a Blaise survey. There are many alternatives, however some are not compatible across operating systems, particularly between Windows NT/2000 and Windows 98/95. While some solutions may function correctly, the user interface may be awkward or unappealing. This paper will review some of the less desirable methods and conclude with a recommended solution.

First, it is worthwhile to quickly review some less desirable shelling methods. These methods usually work, however each one has a downside. The less desirable solutions covered in this paper will involve DOS batch files and Windows API calls. Sample Visual Basic code for these solutions will be reviewed.

After reviewing some less desirable methods, this paper will reveal a simple and effective solution that does not have the problems explained in the previous section. Sample Visual Basic code for this solution will be reviewed. The Visual Basic code will first call a Manipula program to pass data from a Microsoft Access database into a Blaise database. That same code will then run a Blaise Computer Assisted Interview (CAI). Upon terminating the CAI, Visual Basic will then call a Manipula program to determine the CAI completion status (e.g. complete, partial complete, etc.).

This solution will not require DOS batch files or Windows API calls and all calls to Manipula.exe and Dep.exe will be made from within the Visual Basic code. In addition, the code will ensure that all processing occurs one request at a time in order (synchronously) and will be compatible between the Windows NT/2000 and Windows 98/95 operating systems.

Less Desirable Shelling Methods

DOS Batch Files

DOS Batch files may be used to synchronously process several requests. Calling processes **synchronously** will ensure that each process will not start until the preceding process is finished. For example, you might need to run a manipula program to add or update a Blaise record before running the datamodel. Then you might want to export data from that same datamodel after closing it. Here is a sample DOS batch file that illustrates the above.

```
rem THIS IS A SAMPLE BATCH FILE THAT ADDS A RECORD TO
rem A BLAISE DATAMODEL VIA addid.man THEN RUNS THE DATAMODEL (cati_1).
rem AFTER THE DATAMODEL IS CLOSED, IT RUNS result.man TO
rem EXPORT RESULTS FROM THE DATAMODEL TO AN ASCII FILE.
 @echo off
 cls
 echo Initializing interview.....

rem SET WORKING DRIVE
 K:

rem SET WORKING DIRECTORY
 cd\projects\cati

rem 'Start /w' IS REQUIRED WHEN NEEDING TO RUN WINDOWS PROGRAMS SYNCHRONOUSLY.
rem WITHOUT START /W, EACH CALL TO A WINDOWS PROGRAM WILL RUN AT THE SAME TIME INSTEAD
rem INSTEAD OF IN ORDER.
rem 'Start /w' IS NOT NEEDED TO RUN DOS PROGRAMS SYNCHRONOUSLY.

rem RUN THE MANIPULA PROGRAM addid.man TO ADD/UPDATE A RECORD.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO MANIPULA.
 Start /w k:\blaise\windows\vER_452b642\manipula.exe addid /q /p%1

rem RUN THE DATAMODEL cati_1 AFTER THE addid.man IS FINISHED.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO DEP.
 Start /w k:\blaise\windows\vER_452b642\dep.exe cati_1 /g /k%1 /x

rem RUN THE MANIPULA PROGRAM result.man TO GET CERTAIN RESULTS EXPORTED FROM
rem THE cati_1 DATAMODEL AFTER EXITING dep.exe.
rem '%1' IS A PARAMETER SENT INTO THE BATCH FILE AND THEN SENT INTO MANIPULA.
 Start /w k:\blaise\windows\vER_452b642\manipula.exe result /q /p%1

rem NOTE THAT WE USE Start /w ON THE LAST CALL TOO.  THIS DOES NOT MAKE SENSE
rem IF YOU ARE ONLY INTERESTED IN MAKING THE CALLS IN THE DOS BATCH FILE
rem SYNCHRONOUS.  HOWEVER, IF YOU WANT TO ALSO SYNCHRONOUSLY CALL THE DOS
rem BATCH FILE WITHIN ANOTHER PROGRAM (I.E. VISUAL BASIC), YOU WANT THE
rem DOS BATCH FILE TO REMAIN OPEN UNTIL ALL PROCESSES INSIDE IT ARE FINISHED.

rem CLOSE THE BATCH FILE WINDOW
 Exit
```

The downsides of this approach are that it creates another link in the programming chain that further complicates the programmer's job, DOS Batch files may not be supported in the future, and DOS may pop-up in a window that confuses the user. It also raises code security concerns unless the programmer changes the properties of the DOS batch file to prohibit users from editing or deleting it.
You may call a DOS batch file within a Visual Basic project with the Shell command.
Example:

```
Dim RetVal
RetVal = Shell("C:\Sample\Sample.bat", 1)
```

According to Microsoft, *"By default, the Shell function runs other programs **asynchronously**. This means that a program started with Shell might not finish executing before the statements following the*

Shell function are executed." This restricts your control within Visual Basic. Other means are necessary if you need to run several commands synchronously. The following Windows API methods can solve that problem. However an even better solution will be covered later.

Windows API Functions

In a Visual Basic Project, you might have code like the following examples to synchronously shell to external programs. This code incorporates Windows API functions. On first glance this code is complicated and not for the novice programmer.

```
Option Explicit

Public Const GW_HWNDFIRST = 0
Public Const GW_HWNDLAST = 1
Public Const GW_HWNDNEXT = 2
Public Const GW_HWNDPREV = 3

Public Const GW_CHILD = 5
Public Const GW_MAX = 5

Global Const NORMAL_PRIORITY_CLASS = &H20&
Global Const INFINITE = -1&

Public Const STILL_ACTIVE = &H103
Public Const PROCESS_QUERY_INFORMATION = &H400

Public Const GW_OWNER = 4
Public Const GWL_STYLE = -16
Public Const WS_DISABLED = &H8000000
Public Const WS_CANCELMODE = &H1F
Public Const WM_CLOSE = &H10

Public glCurrentHwnd

Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessID As Long
    dwThreadID As Long
End Type

'THESE DECLARE FUNCTIONS ARE ALL WINDOWS API CALLS
'CONFUSING AND HARD TO USE FOR NOVICE PROGRAMMERS.
'ALSO REQUIRES ADDITIONAL SYSTEM RESOURCES
Declare Function OpenProcess Lib "kernel32" _
   (ByVal dwDesiredAccess As Long, ByVal bInheritHandle _
   As Long, ByVal dwProcessID As Long) As Long
Declare Function GetExitCodeProcess Lib "kernel32" _
```

```
   (ByVal hProcess As Long, lpExitCode As Long) As Long
Declare Function CloseHandle Lib "kernel32" (hObject As Long) _
    As Boolean
Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
    hHandle As Long, ByVal dwMilliseconds As Long) As Long
Declare Function CreateProcessA Lib "kernel32" _
   (ByVal lpApplicationName As Long, ByVal lpCommandLine As _
   String, ByVal lpProcessAttributes As Long, ByVal _
   lpThreadAttributes As Long, ByVal bInheritHandles As _
   Long, ByVal dwCreationFlags As Long, ByVal _
   lpEnvironment As Long, ByVal _
   lpCurrentDirectory As Long, lpStartupInfo As STARTUPINFO, _
   lpProcessInformation As PROCESS_INFORMATION) As Long
Declare Function IsWindow Lib "user32" (ByVal hwnd As Long) _
    As Long
Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, _
    ByVal wCmd As Long) As Long
Declare Function PostMessage Lib "user32" Alias _
    "PostMessageA" (ByVal hwnd As Long, ByVal wMsg _
    As Long, ByVal wParam _
    As Long, ByVal lParam As Long) As Long
Declare Function GetWindowLong Lib "user32" Alias _
    "GetWindowLongA" (ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
Declare Function GetParent Lib "user32" (ByVal hwnd _
As Long) As Long
Declare Function GetWindowTextLength Lib "user32" Alias _
    "GetWindowTextLengthA" (ByVal hwnd As Long) As Long
Declare Function GetWindowText Lib "user32" Alias _
    "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString _
    As String, ByVal cch As Long) As Long

Dim strAppToLaunch As String

'USE SHELLANDCONTINUE TO ALLOW THE VISUAL BASIC
'TRACKING SYSTEM TO STILL BE USED WHILE RUNNING THE SPECIFIED PROCESS.
'THIS PROCESS STILL KEEPS TRACK OF WHEN THE CALLED PROCESS IS
'DONE PROCESSING.  WHEN THAT HAPPENS, THE CODE CONTINUES WITHIN
'THE SHELLANDCONTINUE FUNCTION.  SO, IT REALLY DOES STOP EXECUTION
'WITHIN THAT FUNCTION, BUT STILL ALLOWS USER TO DO OTHER THINGS IN THE
'TRACKING SYSTEM.

Call ShellAndContinue(strAppToLaunch)

'OR YOU CAN USE SHELLANDWAIT TO CAUSE THE VISUAL BASIC TRACKING SYSTEM
'TO COMPLETELY LOCKUP WHILE RUNNING THE SPECIFIED PROCESS.
'THIS PROHIBITS THE USER FROM DOING ANYTHING IN THE TRACKING SYSTEM
'WHILE THE PROCESS IS STILL RUNNING.  THIS IS CUMBERSOME AND AT TIMES
'CONFUSING TO THE END USER, BUT MAKES LIFE EASIER FOR THE PROGRAMMER
'BEACAUSE THEY DON'T HAVE TO WORRY ABOUT CONTROLLING USER EVENTS WHILE
'RUNNING THE EXTERNAL PROCESS.

Call ShellAndWait(strAppToLaunch)

....

Sub ShellAndContinue(ByVal AppToRun As String)
    On Error GoTo ErrorRoutineErr

    Dim hProcess As Long
    Dim RetVal As Long
    Dim Msg, Style, Title, Response 'msgbox variables

    'The next line launches AppToRun,
    'captures process ID
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, 1, _
    Shell(AppToRun, vbNormalFocus))
```

```
    Do
        'Get the status of the process
        GetExitCodeProcess hProcess, RetVal
        DoEvents
    'Loop while the process is active
    Loop While RetVal = STILL_ACTIVE

    'define message
    Msg = AppToRun & " Terminated by user"
    'define buttons
    Style = vbOKOnly + vbInformation
    'define title
    Title = "Termination Notice"
    'display message
    Response = MsgBox(Msg, Style, Title)

ErrorRoutineResume:
    Exit Sub
ErrorRoutineErr:
    MsgBox "AppShell.Form1.ShellAndContinue" & Err & Error
End Sub

Public Sub ShellAndWait(AppToRun)
    On Error GoTo ErrorRoutineErr

    Dim NameOfProc As PROCESS_INFORMATION
    Dim NameStart As STARTUPINFO
    Dim rc As Long

    NameStart.cb = Len(NameStart)
    rc = CreateProcessA(0&, AppToRun, 0&, 0&, 1&, _
       NORMAL_PRIORITY_CLASS, _
       0&, 0&, NameStart, NameOfProc)
    rc = WaitForSingleObject(NameOfProc.hProcess, INFINITE)
    rc = CloseHandle(NameOfProc.hProcess)

ErrorRoutineResume:
    Exit Sub
ErrorRoutineErr:
    MsgBox "AppShell.Form1.ShellAndWait" & Err & Error
    Resume Next

End Sub
```

The problems with the above code are that it is confusing and resource intensive. Also, some of it cannot be used to call MANIPULA.EXE or DEP.EXE. Specifically, it seems that the ShellAndWait method completely locks up the application when attempting to call MANIPULA or DEP. This happens when the WaitForSingleObject function is called. This problem forces the programmer to wrap up the MANIPULA and DEP calls inside a DOS batch file or some other kind of scripting file and then call that script file with the above procedures. This starts to be an entangled mess that is hard to debug and confusing for programmers. In case you are wondering, the ShellAndWait function works for other applications like NOTEPAD.EXE. The problem seems to be Blaise related.

RECOMMENDED METHOD FOR SYNCHRONOUS SHELLING

There are other ways to synchronously shell from Visual Basic to MANIPULA and/or DEP, however we should move on to this paper's recommended way to do the same thing. The heart of the recommended code replaces all of the code shown above. This code is bolded inside the class 'clsProcess'. It's amazing how brief this code is compared to the ShellAndWait and ShellAndContinue examples.

Here is the code that replaces the code above:

```
'FOR STORING THE MANIPULA/DEP COMMAND STRING
Dim strCommand as String
'FOR CONTROLLED SHELLING TO MANIPULA/DEP
Dim WshShell As IWshShell

'CREATE THE WSHSHELL OBJECT
Set WshShell = CreateObject("Wscript.Shell")

'CREATE THE VARIABLE TO RECEIVE THE RETURN VALUE FROM WSHSHELL
Dim lngAppReturnValue As Long

'RUN THE MANIPULA/DEP COMMAND STRING, TELLING IT TO USE A MAXIMIZED WINDOW (3)
'AND SET THE BOOLEAN PARAMETER TO 'TRUE' SO WSHSHELL WILL WAIT UNTIL THE PROCESS IS FINISHED
'BEFORE PROCEEDING TO THE NEXT LINE OF CODE.
lngAppReturnValue = WshShell.Run(strCommand, 3, True)
```

Microsoft's Windows Script Host (WshShell) object exposes some of the common shell functionalities of Microsoft Windows, making it easy to create shortcuts, access the environment variables, run applications, access system registry, and more. The WshShell object is not instantiated automatically upon script execution, hence it must be instantiated explicitly using CreateObject before it can be used.

The following code is organized into Class modules. A programmer can instantiate the code as an object and send the required parameters to the appropriate methods. This frees the programmer from having to repeatedly delve into more complicated code. It also allows for easier upgrade of code, because the code is only repeated once. Specifically, these classes shield the front-end Visual Basic programmer who doesn't know much about Blaise from having to get into the details of how to call a Manipula program or datamodel.

It's important to note that the classes specified below are not required. The additional code was added to make life easier for the 'front-end' programmers who design the end user's Visual Basic forms. If you are not interested in using these classes, you can simply use the few lines of code bolded at the beginning of this section.

```
'Programmer Notes:
'Microsoft® Windows® Script Host
'ACCORDING TO MICROSOFT:
'object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
'STRCOMMAND =
'  STRING VALUE INDICATING THE COMMAND LINE YOU WANT TO RUN.
'  YOU MUST INCLUDE ANY PARAMETERS YOU WANT TO PASS TO THE EXECUTABLE FILE.
'INTWINDOWSTYLE (OPTIONAL) =
'  INTEGER VALUE INDICATING THE APPEARANCE OF THE PROGRAM'S WINDOW. NOTE THAT NOT ALL '
'  PROGRAMS MAKE USE OF THIS INFORMATION.
'BWAITONRETURN (OPTIONAL) =
'  BOOLEAN VALUE INDICATING WHETHER THE SCRIPT SHOULD WAIT FOR THE PROGRAM TO FINISH
'  EXECUTING BEFORE CONTINUING TO THE NEXT STATEMENT IN YOUR SCRIPT. IF SET TO TRUE,
'  SCRIPT EXECUTION HALTS UNTIL THE PROGRAM FINISHES, AND RUN RETURNS ANY ERROR CODE
'  RETURNED BY THE PROGRAM (0 FOR SUCCESS). IF SET TO FALSE (THE DEFAULT), THE RUN METHOD
'  RETURNS IMMEDIATELY AFTER STARTING THE PROGRAM, AUTOMATICALLY RETURNING 0 (NOT TO BE
'  INTERPRETED AS AN ERROR CODE).
'
'REMARKS
'  THE RUN METHOD RETURNS AN INTEGER. THE RUN METHOD STARTS A PROGRAM RUNNING IN A NEW
'  WINDOWS PROCESS. YOU CAN HAVE YOUR SCRIPT WAIT FOR THE PROGRAM TO FINISH EXECUTION
'  BEFORE CONTINUING. THIS ALLOWS YOU TO RUN SCRIPTS AND PROGRAMS SYNCHRONOUSLY.
'  ENVIRONMENT VARIABLES WITHIN THE ARGUMENT STRCOMMAND ARE AUTOMATICALLY EXPANDED.
'  IF A FILE TYPE HAS BEEN PROPERLY REGISTERED TO A PARTICULAR PROGRAM, CALLING RUN ON A
'  FILE OF THAT TYPE EXECUTES THE PROGRAM. FOR EXAMPLE, IF WORD IS INSTALLED ON YOUR
'  COMPUTER SYSTEM, CALLING RUN ON A *.DOC FILE STARTS WORD AND LOADS THE DOCUMENT.
'  THE FOLLOWING TABLE LISTS THE AVAILABLE SETTINGS FOR INTWINDOWSTYLE.
'
'INTWINDOWSTYLE DESCRIPTION
'0 HIDES THE WINDOW AND ACTIVATES ANOTHER WINDOW.
'1 ACTIVATES AND DISPLAYS A WINDOW. IF THE WINDOW IS MINIMIZED OR MAXIMIZED,
'  THE SYSTEM RESTORES IT TO ITS ORIGINAL SIZE AND POSITION. AN APPLICATION
'  SHOULD SPECIFY THIS FLAG WHEN DISPLAYING THE WINDOW FOR THE FIRST TIME.
'2 ACTIVATES THE WINDOW AND DISPLAYS IT AS A MINIMIZED WINDOW.
'3 ACTIVATES THE WINDOW AND DISPLAYS IT AS A MAXIMIZED WINDOW.
'4 DISPLAYS A WINDOW IN ITS MOST RECENT SIZE AND POSITION. THE ACTIVE WINDOW REMAINS ACTIVE.
'5 ACTIVATES THE WINDOW AND DISPLAYS IT IN ITS CURRENT SIZE AND POSITION.
'6 MINIMIZES THE SPECIFIED WINDOW AND ACTIVATES THE NEXT TOP-LEVEL WINDOW IN THE Z ORDER.
'7 DISPLAYS THE WINDOW AS A MINIMIZED WINDOW. THE ACTIVE WINDOW REMAINS ACTIVE.
'8 DISPLAYS THE WINDOW IN ITS CURRENT STATE. THE ACTIVE WINDOW REMAINS ACTIVE.
'9 ACTIVATES AND DISPLAYS THE WINDOW. IF THE WINDOW IS MINIMIZED OR MAXIMIZED,
'  THE SYSTEM RESTORES IT TO ITS ORIGINAL SIZE AND POSITION. AN APPLICATION SHOULD
'  SPECIFY THIS FLAG WHEN RESTORING A MINIMIZED WINDOW.
'10 SETS THE SHOW-STATE BASED ON THE STATE OF THE PROGRAM THAT STARTED THE APPLICATION.
```

```
'*********************************************************************
'CLASS: clsInterview
'PURPOSE:  Allow easy front end management of external Blaise interviews
'*********************************************************************
Option Explicit

'USER DEFINED OBJECT FOR RUNNING APPLICATION PROCESSES
'OUTSIDE OF THE VISUAL BASIC APPLICATION
Private moProcess As New clsProcess
'STRING HOLDING LOCATION OF THE MANIPULA.EXE TO USE
Const MANIPULA_EXE = "K:\PROJECTS\CATI\BLAISE\WINDOWS\Ver_424B518\MANIPULA.EXE"
'STRING HOLDING LOCATION OF THE DEP.EXE TO USE
Const DEP_EXE = "K:\PROJECTS\CATI\BLAISE\WINDOWS\Ver_424B518\DEP.EXE"

'*********************************************************************
'PURPOSE:  TO START A SPECIFIED INTERVIEW NUMBER
'INPUTS:  INTERVIEW #, SUBJECT ID, DIRECTORY OF INPUT FILES (OPTIONAL),
'         DATA STRING BEING PASSED TO INPUT FILE (OPTIONAL)
'RETURNS:  BOOLEAN TRUE MEANS SUCCESSFULLY STARTED INTERVIEW
'          BOOLEAN FALSE MEANS ATTEMPT TO START INTERVIEW FAILED
'PROGRAMMER:  JOHN CASHWELL
'LAST UPDATED:  7.31.2001
'*********************************************************************
Public Function StartInterview(intInterviewNumber As Integer, strID As String, Optional
strInputDirectory As String, Optional strData As String) As Boolean
On Error GoTo ERR_HANDLER

    StartInterview = False

    Select Case intInterviewNumber
    Case 1
        'WRITE THE PARAMETER strData TO AN ASCII FILE WHICH WILL BE IMPORTED INTO BLAISE
        Open strInputDirectory & "\" & Trim$(strID) & ".in" For Output As #1
        Print #1, strData
        Close #1

        'RUN MANIPULA TO INITIALIZE CASE IN BLAISE
        If moProcess.RunManipulaProcess( _
            MANIPULA_EXE, _
            "START.MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
            True, False, _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\INPUT\" & Trim$(strID) & ".in" _
            , , Trim$(strID)) = False Then
                Exit Function
        End If
        'RUN CATI INTERVIEW
        If moProcess.RunDEPProcess(DEP_EXE, "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
                    "INTERVIEW", strID, True, " ") = False Then
            Exit Function
        End If

        'RUN MANIPULA TO RETURN RESULTS
        If moProcess.RunManipulaProcess( _
            MANIPULA_EXE, "END.MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\MAN", _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE", _
            True, False, , _
            "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\OUTPUT\" & Trim$(strID) & ".OUT", _
            Trim$(strID)) = False Then
            Exit Function
        End If
        'SUCCESSFUL COMPLETION OF ALL TASKS
        StartInterview = True
    Case 2
        'RESERVED FOR FOLLOW-UP INTERVIEWS
```

```
        'CASE 3 ETC..
    End Select

    Exit Function
ERR_HANDLER:
    Call ErrorHandler
End Function

'*********************************************************************
'PURPOSE:  TO READ INTERVIEW RESULT FROM A SPECIFIED FILE
'INPUTS:   DIRECTORY OF INPUT FILE, NAME OF INPUT FILE
'RETURNS:  RESULT STRING (i.e. C=COMPLETED, P=PARTIAL, etc..)
'PROGRAMMER:  JOHN CASHWELL
'LAST UPDATED:  7.31.2001
'*********************************************************************
Public Function GetResult(strDirectory As String, strFileName As String) As String
On Error GoTo ERR_HANDLER
    Dim strRow As String

    If Dir(strDirectory & "\" & strFileName) = "" Then
        MsgBox "The result file '" & strDirectory & "\" & strFileName & "' " & _
                "does not exist.  Please notify programmer."
        Exit Function
    End If

    Open strDirectory & "\" & strFileName For Input As #1

    Do Until EOF(1)
    Line Input #1, strRow
    Loop

    GetResult = strRow

    GoTo CLEAN_UP
ERR_HANDLER:
    Call ErrorHandler
    GoTo CLEAN_UP
CLEAN_UP:
    Close #1
End Function
```

```
'*********************************************************************
'CLASS: clsProcess
'PURPOSE:  To control Manipula and DEP processes.
'*********************************************************************
Option Explicit

'*********************************************************************
'PURPOSE:  TO EASILY RUN MANIPULA PROGRAMS AND CONTROL THE ORDER
'          OF EVENTS BETWEEN THE TRACKING SYSTEM AND MANIPULA
'INPUTS:   MANIPULA.EXE'S PATH AND FILE NAME,
'          MANIPULA PROGRAM'S NAME,
'          MANIPULA PROGRAM'S DIRECTORY PATH,
'          DATAMODEL'S DIRECTORY PATH,
'          BOOLEAN FOR WHETHER TO RUN MANIPULA IN QUIET MODE,
'          BOOLEAN FOR WHETHER TO VIEW MANIPULA RESULTS WHEN DONE PROCESSING
'            (Quite Mode Must = False To Have View Results = True),
'          MANIPULA INPUT FILE PATH AND FILE NAME (OPTIONAL),
'          MANIPULA OUTPUT FILE PATH AND FILE NAME (OPTIONAL),
'          PARAMETERS TO SEND INTO MANIPULA PROGRAM (OPTIONAL),
'          ANY ADDITIONAL OPTIONS (OPTIONAL)
'RETURNS:  BOOLEAN TRUE IF SUCCESSFULLY RUNS MANIPULA PROCESS
'PROGRAMMER:  JOHN CASHWELL
'LAST UPDATED:  7.31.2001
'*********************************************************************
Public Function RunManipulaProcess( _
        strManipula_EXE_Path_And_File_Name, _
        strManipula_Program_Name As String, _
        strManipula_Program_Path As String, _
        strDatamodel_Path As String, _
        blnQuiet As Boolean, _
        blnStop_And_View_Results As Boolean, _
        Optional strInput_File_Path_And_Name As String, _
        Optional strOutput_File_Path_And_Name As String, _
        Optional strParameters As String, _
        Optional strAdditional_Options As String) As Boolean
On Error GoTo ERR_HANDLER

    'IN ORDER TO USE THE IWSHSHELL OBJECT, YOU NEED TO HAVE
    'A REFERENCE IN VISUAL BASIC TO:  WSHOM.OCX
    'IT IS CALLED 'WINDOWS SCRIPT HOST OBJECT MODEL'

    Dim WshShell As IWshShell 'FOR CONTROLLED SHELLING TO MANIPULA
    Dim strDriveLetter As String 'TO SET WORKING DRIVE
    Dim strCommand As String 'TO BUILD THE FULL MANIPULA COMMAND STRING

    'GET DRIVER LETTER
    strDriveLetter = Left(strManipula_Program_Path, 1)

    'INITIALIZE RETURN VALUE TO fALSE
    RunManipulaProcess = False

    'CREATE THE WSHSHELL OBJECT AS WINDOWS HOST SCRIPT OBJECT
    'THIS OBJECT IS USED TO CONTROL SHELLING TO EXTERNAL
    'APPLICATION LIKE BLAISE
    Set WshShell = CreateObject("Wscript.Shell")

    'CHANGE DIRECTORY TO WHERE THE MANIPULA PROGRAM RESIDES
    If Dir(strManipula_Program_Path, vbDirectory) = "" Then
        MsgBox "The directory specified for " & strManipula_Program_Name & _
                " does not exist.  Process cancelled."
        Exit Function
    End If

    'CHANGE DRIVE AND DIRECTORY TO FOLDER THE MANIPULA PROGRAMS RESIDE
    ChDrive strDriveLetter
    'CHANGE THE DIRECTORY
    ChDir strManipula_Program_Path
```

```
    'BUILD THE COMMAND STRING FROM THE OPTIONS PASSED IN TO FUNCTION
    strCommand = strManipula_EXE_Path_And_File_Name
    strCommand = strCommand & " " & strManipula_Program_Name
    strCommand = strCommand & " /F" & strDatamodel_Path
    strCommand = strCommand & IIf(blnQuiet, " /Q", "")
    strCommand = strCommand & IIf(blnStop_And_View_Results, " /A", "")
    strCommand = strCommand & IIf(Trim$(strInput_File_Path_And_Name) = "", "", " /I" &
                 strInput_File_Path_And_Name)
    strCommand = strCommand & IIf(Trim$(strOutput_File_Path_And_Name) = "", "", " /O" &
                 strOutput_File_Path_And_Name)
    strCommand = strCommand & IIf(Trim$(strParameters) = "", "", " /P" & strParameters)
    strCommand = strCommand & " " & strAdditional_Options

    'NOTE: THE 'HALT' MANIPULA COMMAND WILL MAKE THE MANIPULA
    'PROGRAM APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
    'BACK FROM BLAISE ABOUT WHAT HAPPENED.

    'NOTE: lngAppReturnValue HAS A VALUE GREATER THAN 0 IF WSHSHELL
    'IS WAITING FOR THE APPLICATION TO END AND THE APPLICATION HAS
    'FAILED IN SOME WAY - USE lngAppReturnValue TO GET RETURN VALUE FROM WSHSHELL
    Dim lngAppReturnValue As Long

    lngAppReturnValue = WshShell.Run(strCommand, 3, True)

    If lngAppReturnValue > 0 Then
        'CODE FOR FAILURE OF COMMAND CALL
        MsgBox "The Manipula call: '" & strCommand & "' failed for some reason.  " & _
               "Please notify programmer."
        Exit Function
    End If

    RunManipulaProcess = True
    Exit Function
ERR_HANDLER:
    Call ErrorHandler

End Function
```

```
'*********************************************************************
'PURPOSE:  TO EASILY RUN DATAMODELS (DEP.EXE) AND CONTROL THE ORDER
'          OF EVENTS BETWEEN THE TRACKING SYSTEM AND DEP
'INPUTS:   DEP.EXE'S PATH AND FILE NAME,
'          DATAMODEL'S DIRECTORY PATH,
'          DATAMODEL'S FILE NAME,
'          SUBJECT ID TO USE IN DATAMODEL,
'          BOOLEAN FOR WHETHER TO EXIT DEP WHEN FINISHED WITH
'             SPECIFIED ID'S INTERVIEW,
'          MENU FILE NAME IF USING ONE (OPTIONAL)
'RETURNS:  BOOLEAN TRUE IF SUCCESSFULLY RUNS DEP PROCESS
'PROGRAMMER:  JOHN CASHWELL
'LAST UPDATED:  7.31.2001
'*********************************************************************
Public Function RunDEPProcess( _
    strDEP_EXE_Path_And_File_Name As String, _
    strDatamodel_Path As String, _
    strDatamodel_File_Name As String, _
    strID As String, _
    blnExit_Blaise_When_Done As Boolean, _
    Optional strBlaise_Menu_File_Name As String) As Boolean

On Error GoTo ERR_HANDLER

    'WSHSHELL OBJECT
    'object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
    'SEE FUNCTION RunManipulaProcess FOR DIRECTIONS
    Dim WshShell As IWshShell 'FOR CONTROLLED SHELLING TO DEP
    Dim strDriveLetter As String 'TO SET WORKING DRIVE
    Dim strCommand As String 'TO BUILD THE FULL DEP COMMAND STRING

    'GET DRIVER LETTER
    strDriveLetter = Left(strDatamodel_Path, 1)

    'INITIALIZE RETURN VALUE TO FALSE
    RunDEPProcess = False

    'CREATE THE WSHSHELL OBJECT
    Set WshShell = CreateObject("Wscript.Shell")

    'CHANGE DIRECTORY TO WHERE THE MAN AND BLAISE DATA RESIDES
    If Dir(strDatamodel_Path, vbDirectory) = "" Then
        MsgBox "The directory specified for " & strCommand & _
                " does not exist.  Process cancelled."
        Exit Function
    End If
    'CHANGE DRIVE AND DIRECTORY TO FOLDER THE MAN AND BLAISE DATA RESIDES
    ChDrive strDriveLetter
    'CHANGE THE DIRECTORY
    ChDir strDatamodel_Path

    'WSHSHELL OBJECT
    'object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
    'SEE FUNCTION RunManipulaProcess FOR DIRECTIONS
    'NOTE: THE HALT COMMAND IN MANIPULA WILL MAKE THE MANIPULA
    'APPLICATION APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
    'BACK FROM BLAISE ABOUT WHAT HAPPENED.

    'BUILD THE COMMAND STRING FROM THE OPTIONS PASSED IN TO FUNCTION
    strCommand = strDEP_EXE_Path_And_File_Name
    strCommand = strCommand & " " & strDatamodel_File_Name
    strCommand = strCommand & IIf(Trim$(strID) = "", "", " /G /K" & Trim$(strID))
    strCommand = strCommand & IIf(blnExit_Blaise_When_Done, " /X", "")
    strCommand = strCommand & IIf(Trim$(strBlaise_Menu_File_Name) = "", "", " /M" &
                 strBlaise_Menu_File_Name)
```

```
    'NOTE: THE 'HALT' MANIPULA COMMAND WILL MAKE THE MANIPULA
    'PROGRAM APPEAR TO FAIL GIVING US A WAY TO GET A MESSAGE
    'BACK FROM BLAISE ABOUT WHAT HAPPENED.

    'NOTE: lngAppReturnValue HAS A VALUE GREATER THAN 0 IF WSHSHELL
    'IS WAITING FOR THE APPLICATION TO END AND THE APPLICATION HAS
    'FAILED IN SOME WAY - USE lngAppReturnValue TO GET RETURN VALUE FROM WSHSHELL

    Dim lngAppReturnValue As Long

    lngAppReturnValue = WshShell.Run(strCommand, 3, True)

    If lngAppReturnValue > 0 Then
        'CODE FOR FAILURE OF COMMAND CALL
        MsgBox "The call to " & strCommand & " failed for some reason.  " & _
                "Please notify programmer."
        Exit Function
    End If

    RunDEPProcess = True
    Exit Function
ERR_HANDLER:
    Call ErrorHandler

End Function
```

Here is some sample front-end code showing the above classes in action.

```
'THIS CODE RESIDES UNDER A VISUAL BASIC FORM WHICH DISPLAYS
'A DATAGRID OF SUBJECTS TO BE INTERVIEWED AND A BUTTON TO START
'AN INTERVIEW.  THE DATAGRID INCLUDES SUBJECT IDS, NAMES, AND GENDERS

Option Explicit

'CLSINTERVIEW OBJECT CREATED TO RUN BLAISE INTERVIEWS
Private moInterview As New clsInterview
'MSTRID IS A USED TO STORE THE CURRENT ID NUMBER SELECT IN A DATAGRID
Private mstrID As String

'THIS ROUTINE OCCURS WHEN THE SUBJECT CLICKS THE 'START INTERVIEW' BUTTON
Private Sub cmdInterview_Click()
On Error GoTo ERR_HANDLER

    'STRIMPORTSTRING IS FOR CREATING THE DATA STRING THAT WILL BE
    'PASSED TO MANIPULA (SUBJECTID+NAME+GENDER)
    Dim strImportString As String

    'MSTRID IS FILLED WITH COLUMN DATA FROM THE FORM'S DATAGRID (DGSUBJECTS)
    mstrID = Trim(dgSubjects.Columns(0).Text)

    If mstrID = "" Then
        MsgBox "Please select a subject."
        Exit Sub
    End If

    'WHEN THE END USER CLICKS THE START INTERVIEW BUTTON,
    'DISABLE THE FORM BUTTONS FOR STARTING THE INTERVIEW AND EXITING THE FORM
    cmdInterview.Enabled = False
    cmdClose.Enabled = False
    With dgSubjects
        strImportString = mstrID 'StudyID
        strImportString = strImportString & "," & .Columns(1).Text 'fname
        strImportString = strImportString & "," & .Columns(2).Text 'lname
        strImportString = strImportString & "," & .Columns(3).Text 'gender
    End With

    'CALL THE clsInterview METHOD DO RUN INTERVIEW #1, PASSING IN THE ID
    'INPUT FILE's PATH (FYI, THE INPUT FILE IS EXPECTED TO BE NAMED WITH THE SUBJECTID
    'i.e. 1000.in), AND THE DATA STRING TO BE IMPORTED INTO MANIPULA

    If moInterview.StartInterview(1, mstrID,
       "K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\INPUT", strImportString) Then

            'IF THE STARTINTERVIEW METHOD RETURN 'TRUE' THEN RUN CLSINTERVIEW'S
            'GETRESULT METHOD AND STORE THE RETURN VALUE IN THIS FORM'S DATAGRID.
            'IT SHOULD RETURN 'C' FOR COMPLETES AND 'P' FOR PARTIAL COMPLETES.
            'GETRESULT EXPECTS YOU TO PASS IN THE OUTPUT FILE DIRECTORY AND FILENAME.
            'THAT OUTPUT FILE IS CREATED BY A MANIPULA PROGRAM (END.MAN) WHICH IS CALLED
            'IN THE CLSINTERVIEW.STARTINTERVIEW METHOD (FYI, THE OUTPUT FILE IS EXPECTED TO BE
             NAMED WITH THE SUBJECTID, i.e. 1000.out)
             dgSubjects.Columns(4).Text =
             moInterview.GetResult("K:\PROJECTS\CATI\CPHRE_TRACKING\BLAISE\OUTPUT", mstrID &
             ".OUT")
             dgSubjects.Refresh
    End If

    'WHEN THE MANIPULA PROGRAMS AND BLAISE INTERVIEW ARE FINISHED PROCESSING,
    'ENABLE THE FORM BUTTONS FOR STARTING THE INTERVIEW AND EXITING THE FORM
    cmdInterview.Enabled = True
    cmdClose.Enabled = True
    Exit Sub
ERR_HANDLER:
```

```
    Call ErrorHandler
End Sub
```

It is worth noting that a programmer can use the HALT command in Manipula programs to prematurely end them when something is not correct. If the Manipula program is called with Microsoft's Windows Host Script, an error level code is automatically returned when a HALT is processed. This allows the Visual Basic programmer to use the error level code to notify the end user that an error was encountered and the interview can be cancelled. Here is a snippet from a Manipula program where the HALT command is used.

```
…
MANIPULATE
  infile.READNEXT

  if (infile.StudyID<>PARAMETER) OR
     (infile.Gender=EMPTY) OR
     (infile.FNAME=EMPTY) OR
     (infile.LNAME=EMPTY) THEN

        HALT

  else
…
```

It is important to understand that Microsoft's Windows Host Script allows the Visual Basic application to be used while it is processing other requests. The only place the code waits for the external process to end is inside the function where the WshShell object is called. This means that the programmer can be assured that the code within the function containing the Windows Host Script object will operate synchronously. However, the programmer must alter the properties of the form as needed to control the user's choices while running a process such as a Blaise datamodel. In the example above, the Start Interview and Exit Form buttons are disabled so the end user cannot exit the form or start another interview while an interview is still being processed. When the interview is finished the button are enabled.

CONCLUSION

In conclusion, there are several ways to synchronously shell between Visual Basic and Blaise. However, Microsoft's Windows Host Script seems to be the best solution. It require only a few lines of code, is easy to understand, works across Windows operating systems when calling Manipula or Dep, frees the end user to continue working in the tracking system, and receives error level codes from Manipula and Dep.