CHAPTER

7

ACTIVE X PROGRAM EXAMPLES

Programming Example:

Simple use of Trio PC Motion ActiveX control in Microsoft Visual BASIC 6.0

Scope This programming example shows how to use the Trio PC Motion ActiveX control in a Microsoft Visual BASIC 6.0 application. It demonstrates how to access system and axis parameters, how to read and write digital I/O and how to perform basic moves.

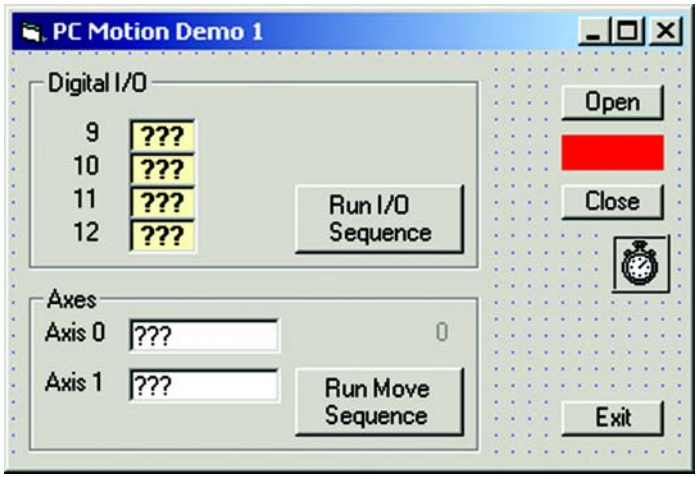
Configuring Visual BASIC

Trio PC Motion ActiveX control must already be installed on the PC.

In order to use the Trio PC Motion ActiveX control Visual BASIC needs to be configured to use the control. To do this, select Project / Components from Visual BASIC's main menu to display the components dialog. The components dialog contains a list of all the controls installed on the PC. Make sure that the check box next to "TrioPC ActiveX Control module" is checked then click on the OK button to close the dialog. The Trio PC Motion ActiveX control icon will appear in the component palette (usually at the bottom, right).

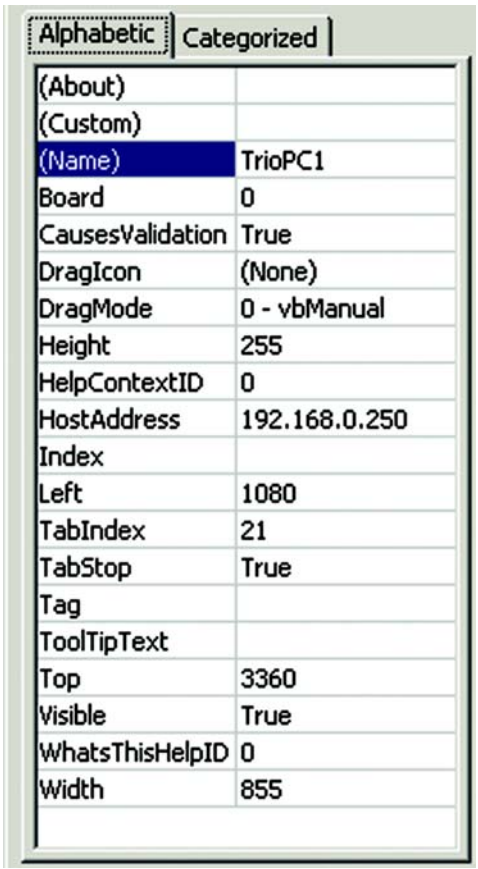
Building the application

Main Form - Visual

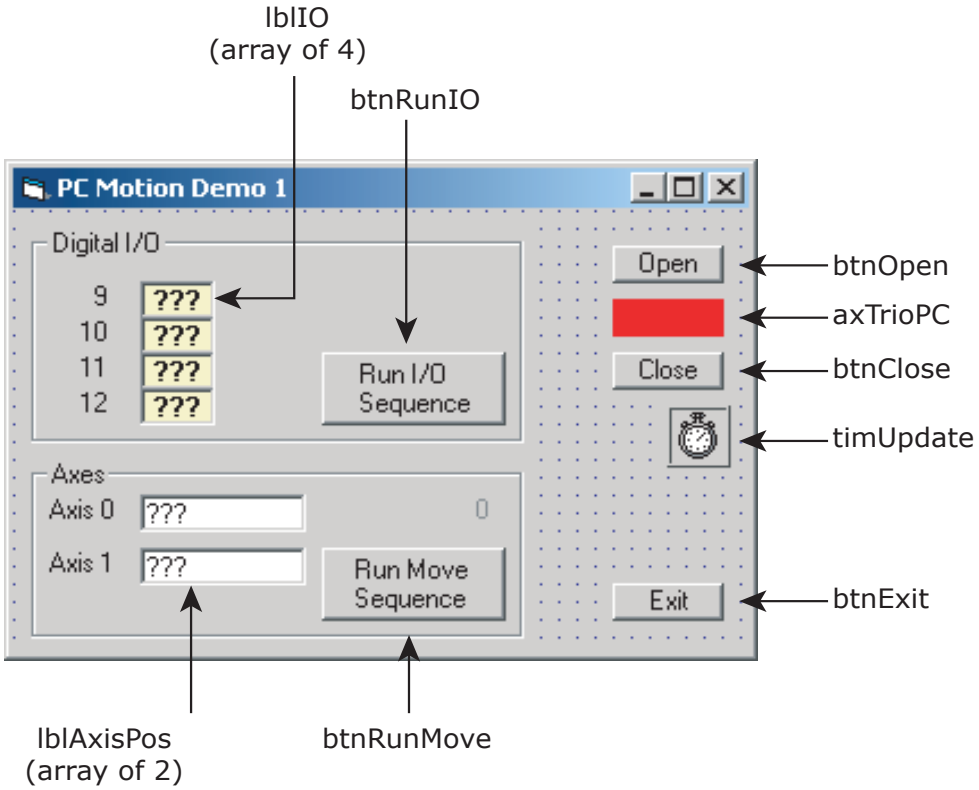


Usually the first stage in building an application is creating the main form (the sample application only has one form). Controls are placed on the form from the component palette the properties of each component being adjusted as appropriate. When the Trio PC Motion ActiveX control is placed on a form it appears as a red, resizable rectangle. The available properties are as shown above.

Only the “Board” and “HostAddress” properties are custom the the control. The “Board” property is only used for a PCI bus connection and the “HostAddress” property is only used for an Ethernet connection. The sample program uses a USB connection so these properties are not used.



The completed main form for the application is shown below, annotated to show the control names used in the example application.



Main Form - Code

The code is best written in stages in order to make testing easier. The starting point for this application was handlers for the "Exit", "Open" and "Close" buttons. The operation of the "Open" and "Close" buttons should be tested before any of the code which accesses the Trio PC Motion component. The handlers for other buttons are then added one at a time, together with any helper functions, and the functionality tested for each group of controls as the Digital I/O group is independent from the Axes (moves) group in this application.

Opening and closing the Trio PC Motion component.

Calling the “Open” should open the connection to the controller. A successful call of the “Open” method should cause the visible Trio PC Motion component to change colour to green (remember to refresh it). The open/closed state of the component can be checked using the “IsOpen” method. Calling the “Close” method will close the connection to the controller. After a successful call of the “Close” the colour of the visible Trio PC Motion component will be red.

The connection should always be closed before the application exits. This is done by putting a conditional call to the “Close” method in the “Form_Unload” routine for the main form in the application. To automatically open the connection a call to the “Open” method can be put in the Form_Load routine of the main form in the example application.

Reading and writing Digital I/O

Digital inputs can be read using the “In” method. This can read the states of a range of inputs. The routine “ReadIO” in this application shows how this is done and how to separate out individual bits from the number returned by the “In” method call.

Digital outputs can be written using the “Op” method. The “Op” method only writes to a single digital output. Writing to a range of outputs can be done by using a program loop such as the FOR loop used in the IO section of the “ProcessStateMachine” routine in the example application.

Reading and writing system parameters

System parameters are read using the “GetVariable” method. The example application demonstrates this when the “WDOG” parameter is read at the beginning of the “InitAxes” routine. The value of the parameter is always returned as a double.

System parameters are written using the “SetVariable” method. The example application demonstrates this when the “WDOG” parameter is written at the beginning and at the end of the “InitAxes” routine.

Reading and writing axis parameters

This uses the same “Get/SetVariable” methods as are used for reading and writing system parameters. The difference is that the parameter read or written is the one from the current base axis. To change the base axis the “Base” method must be used. The “Base” method can specify a single axis or several axes (in an array). If more than one axis is specified then parameter reads and writes use the first axis specified (in element 0 of the array). Examples of this can be seen in the “ReadAxisPositions”, the “ProcessStateMachine” and the “InitAxes” routines in the example

Performing Moves

Moves can be specified using the following methods:

CamBox - Cam shape move linked to another axis

Cam - Cam shape move

Connect - Ratio move linked to another axis

Datum - Datuming move sequence

Forward - Continuous position move in positive direction

Reverse - Continuous position move in reverse direction

MoveAbs – absolute

MoveRel – relative (equivalent to MOVE in Trio BASIC)

MoveCirc – circular

MoveHelical – helical

MoveLink – linked to another axis

MoveModify – modify move end position

The example application shows examples of the “MoveAbs” and “MoveRel” methods in the “ProcessStateMachine” routine. Both of these methods specify an array of move positions/distances. The positions/distances are applied to the axes specified in the last call to the “Base” method. If a “Base” method call specifies axes 1, 2 & 5 (in that order) then a call to “MoveRel” with one axis specified would use axis 1, two axes specified would use axis 1 and axis 2, three axes specified would use axis 1, axis 2 and axis 5.

Moves are loaded into a buffering system for each axis. This is capable of containing two moves, the current move and the next move. If a move is written to an axis which already has a next move specified then the move method called will not return until it is possible for the move to be loaded into the next move position in the move buffer. This can have the effect of hanging the GUI of the calling application unless moves are taken to prevent this. Checking that the “NTYPE” axis parameter is zero on all axes involved in the move is one way of doing this although it may have a side effect of introducing pauses in the motion if the moves are short. The example application demonstrates this method in the “ProcessStateMachine” routine.

Timed operations

Visual BASIC has no means of waiting for a time period therefore any timed operations need to be performed in response to events (usually generated by a timer component). This must be done in order to keep the GUI active during ant "Wait" periods. In the example application the "ProcessStateMachine" method is called by the handler for a timer event. This is to allow the I/O sequence to be performed slowly enough for the counting operation to be seen on the GUI and also to allow the display of axis demand position during the Move sequence.

Program Code

The full source code for the program is shown below:

```
' Demo program for Trio PC Motion ActiveX control
'
'   Trio PC Motion Control version 1.1.0.2 or later is required
'
' Program demonstrates opening and closing connection to controller
' as well as reading and writing axis parameters and I/O states.
'
' I/O and Move sequences are controlled using a state machine run from
' a timer.  This is done to make sure that the values displayed on the PC
' are updated on a regular basis and the PC is not held up waiting for a
' command to complete.  It also allows the I/O sequence to run at a slow
' speed.

Option Explicit                 ' Forces variables to be declared before being used

' Global Variables
Dim g_bRunningIO As Boolean     ' Flag to control running of I/O sequence
Dim g_bRunningMove As Boolean   ' Flag to control running of Move sequence
Dim g_nIOCount As Integer       ' I/O counter used in I/O sequence
Dim g_nMoveNo As Integer        ' Move counter used in Move sequence

' Define connection defaults
Const gk_sDefaultHostAddress As String = "192.168.0.111"
Const gk_nDefaultPciBoard As Integer = 0
Const gk_nDefaultLink As Integer = 0    ' USB link
Const gk_nDefaultMode As Integer = 0    ' Synchronous (Token) mode
Const gk_nMaxAxes As Integer = 24
Const gk_nAxesInUse As Integer = 2

Private Sub Form_Load()
    ' Initialise global variables
    g_bRunningIO = False
    g_bRunningMove = False
    timUpdate.Enabled = True
    g_nIOCount = 0
    g_nMoveNo = 0
    UpdateButtonStates
End Sub

Private Sub btnClose_Click()
    ' Handler for close button - closes connection to controller
```

```
    If axTrioPC.IsOpen(gk_nDefaultMode) Then
        axTrioPC.Close (gk_nDefaultMode)
    End If
    UpdateButtonStates
    Refresh
End Sub

Private Sub btnOpen_Click()
    ' Handler for Open button - opens connection to controller
    Dim bOpen As Boolean

    axTrioPC.HostAddress = gk_sDefaultHostAddress    ' Only needed for ethernet
    axTrioPC.Board = gk_nDefaultPciBoard             ' Only needed for PCI
    bOpen = axTrioPC.Open(gk_nDefaultLink, gk_nDefaultMode)
    UpdateButtonStates
    Refresh

End Sub

Private Sub btnRunIO_Click()
    ' Handler for RunIO button - starts I/O sequence
    g_nIOCount = 0          ' Initialise counter
    ReadIO
    g_bRunningIO = True     ' Indicate I/O sequence running to state machine processor
    UpdateButtonStates
End Sub

Private Sub btnRunMove_Click()
    ' Handler for RunMove button - starts Move sequence
    InitAxes                ' Initialise axis parameters
    g_nMoveNo = 0           ' Initialise counter
    lblMoveNumber.Caption = g_nMoveNo
    lblMoveNumber.Refresh
    ReadAxisPositions
    g_bRunningMove = True   ' Indicate move sequence running to state machine processor
    UpdateButtonStates
End Sub

Private Sub btnExit_Click()
    ' Handler for Exit button
    End
End Sub

Private Sub UpdateButtonStates()
    ' Update button enable states depending on connection state and running sequences
    Dim bOpen As Boolean
```

```
    bOpen = axTrioPC.IsOpen(0)

    btnOpen.Enabled = Not bOpen
    btnClose.Enabled = bOpen And Not (g_bRunningIO Or g_bRunningMove)
    btnRunIO.Enabled = bOpen And Not g_bRunningIO
    btnRunMove.Enabled = bOpen And Not g_bRunningMove
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Make sure link to controller is closed when form closes
    If axTrioPC.IsOpen(gk_nDefaultMode) Then
        axTrioPC.Close (gk_nDefaultMode)
    End If
End Sub

Private Sub ReadIO()
    ' Read Input values used in I/O sequence
    Dim lIO As Long     ' Used to hold value IO value returned from controller
    Dim lMask As Long   ' Used to mask out individual bit values
    Dim nBit As Integer ' Used to count bits
    Dim nMBR As Integer ' Used as dummy for MessageBox return value

    If axTrioPC.IsOpen(gk_nDefaultMode) Then
        If axTrioPC.In(9, 12, lIO) Then     ' Read bits 9 to 12

            ' Cycle through returned bits (0 to 3 equivalent to inputs 9 to 12)
            ' and display bit values in lblIO control array
            lMask = 1
            For nBit = 0 To 3
                If lIO And lMask Then
                    lblIO(nBit).Caption = "On"
                    lblIO(nBit).ForeColor = &HFF&       ' Red
                Else
                    lblIO(nBit).Caption = "Off"
                    lblIO(nBit).ForeColor = &HFF0000    ' Blue
                End If
                lMask = lMask * 2       ' Move mask to next bit
                lblIO(nBit).Refresh     ' Make sure new value is displayed
            Next nBit

        Else
            nMBR = MsgBox("Error reading IO")
        End If
    Else
        nMBR = MsgBox("Connection not open")
    End If

End Sub
```

```
Private Sub ReadAxisPositions()
    ' Read demand positions of axes used in Move sequence
    Dim nAxis As Integer                ' Used for axis number
    Dim nAxes(gk_nMaxAxes) As Integer   ' Array used to set Base axis/axes
    Dim dPosition As Double             ' Used for demand position read from controller
    Dim nMBR As Integer                 ' Used as dummy for MessageBox return value

    ' Initialise Axes array to all zero
    For nAxis = 0 To gk_nMaxAxes - 1
        nAxes(nAxis) = 0
    Next nAxis

    If axTrioPC.IsOpen(gk_nDefaultMode) Then
        ' Scan through all axes in use to read DPOS and show value in lblAxisPos
        '   control array
        For nAxis = 0 To gk_nAxesInUse - 1
            nAxes(0) = nAxis        ' Set up Axes array for Base command
                                    '   (single value in element 0)
            If axTrioPC.Base(1, nAxes) Then
                If axTrioPC.GetVariable("DPOS", dPosition) Then
                    lblAxisPos(nAxis).Caption = Int(dPosition)
                    lblAxisPos(nAxis).Refresh   ' Make sure display is updated
                Else
                    nMBR = MsgBox("Error reading axis positions")
                    nAxis = gk_nAxesInUse   ' Set condition to force loop exit
                End If
            Else
                nMBR = MsgBox("Error setting base for reading axis positions")
                nAxis = gk_nAxesInUse   ' Set condition to force loop exit
            End If
        Next nAxis
    Else
        nMBR = MsgBox("Connection not open")
    End If
 End Sub

Private Sub timUpdate_Timer()
    ' Handler for timer
    If axTrioPC.IsOpen(gk_nDefaultMode) Then
        ProcessStateMachine     ' State machine controls running of I/O and
                                '   Move sequences
    End If
End Sub

Private Sub ProcessStateMachine()
    ' State machine controls running of I/O and Move sequences
```

```
' Called at regular intervals by the timer
Dim nMask As Integer            ' Used to mask out individual I/O bit values
Dim bOK As Boolean              ' Used for value returned by Trio PC commands.
Dim nIO As Integer              ' Used as I/O bit number
Dim nBases(gk_nMaxAxes) As Integer  ' Array used for storing axis numbers used
                                    '   by the Base command
Dim nAxis As Integer            ' Axis number
Dim bOkToMove As Boolean        ' Flag to indicate that it's OK to load moves
Dim dReadVal As Double          ' Used for value returned by GetVariable command
Dim dMoveRef As Double          ' Reference value used in calculation of move
                                '   distances
Dim dMovePos(gk_nMaxAxes) As Double ' Array used for move distances

Const kdRefDistance As Double = 500#    ' Fixed value used in calculating moves
Const knMoves As Integer = 5    ' Number of moves in Move sequence

' IO
If g_bRunningIO Then
    nMask = 1
    ' Set I/O bits 9 to 12 to represent the value in the I/O counter
    For nIO = 0 To 3
        bOK = axTrioPC.Op(nIO + 9, g_nIOCount And nMask)
        nMask = nMask * 2           ' Move mask to next bit
    Next nIO
    ReadIO
    g_nIOCount = g_nIOCount + 1     ' Increment move count
    If g_nIOCount > 15 Then         ' Check for end of move sequence
        g_bRunningIO = False        ' End of sequence so flag it
        UpdateButtonStates
    End If
End If

' Moves
If g_bRunningMove Then
    ' Scan through NTYPE (next move type) on all axes to make sure that moves
    ' can be loaded into the move buffer without holding up the PC.
    bOkToMove = True
    For nAxis = 0 To gk_nAxesInUse - 1
        nBases(0) = nAxis           ' Set single value in Bases array to select
                                    '   axis to read
        bOK = axTrioPC.Base(1, nBases)
        If bOK Then
            bOK = axTrioPC.GetVariable("NTYPE", dReadVal)
            If bOK Then
                If dReadVal <> 0# Then
                    ' Next move still loaded on this axis so mark as not OK
                    '   to load
```

```
                    bOkToMove = False
                End If
            End If
        End If
    Next nAxis

    If bOkToMove Then
        dMoveRef = kdRefDistance * (g_nMoveNo + 1)  ' Move distance is based
                                                    '   on move number

        If g_nMoveNo = 0 Then
            ' First move
            ' Set up Bases for all axes in use and set all moves to zero
            For nAxis = 0 To gk_nAxesInUse - 1
                nBases(nAxis) = nAxis
                dMovePos(nAxis) = 0
            Next nAxis
            bOK = axTrioPC.Base(gk_nAxesInUse, nBases)
            ' First move is absolute to reset position to zero on all used axes
            bOK = axTrioPC.MoveAbs(gk_nAxesInUse, dMovePos)
        ElseIf g_nMoveNo > 0 And g_nMoveNo < knMoves Then
            ' Other moves
            ' Set up bases for all axes and set move distances to
            '   calculated values
            For nAxis = 0 To gk_nAxesInUse - 1
                nBases(nAxis) = nAxis
                dMovePos(nAxis) = dMoveRef * (nAxis + 1)
            Next nAxis
            bOK = axTrioPC.Base(gk_nAxesInUse, nBases)
            ' All other moves are relative
            bOK = axTrioPC.MoveRel(gk_nAxesInUse, dMovePos)
        Else
            ' Terminate move sequence
            g_bRunningMove = False
            UpdateButtonStates
        End If
        lblMoveNumber.Caption = g_nMoveNo
        lblMoveNumber.Refresh
        g_nMoveNo = g_nMoveNo + 1   ' Increment move number
    End If
    ReadAxisPositions
End If
End Sub

Private Sub InitAxes()
    ' Performs axis initialisation for all axes in use
    Dim bOK As Boolean      ' Used as return value for TrioPC motion commands
    Dim nAxis As Integer    ' Used as Axis number
```

```
    Dim nBases(gk_nMaxAxes) As Integer  ' Array used to store axis numbers for
                                        '   Base command
    Dim dReadVal As Double          ' Used for value returned by GetVariable command

    If axTrioPC.IsOpen(gk_nDefaultMode) Then

        ' Modifications should be done with watchdog off
        bOK = axTrioPC.GetVariable("WDOG", dReadVal)   ' Read value should be 0 or 1
        If dReadVal > 0.5 Then
            bOK = axTrioPC.SetVariable("WDOG", 0#)     ' Turn watchdog off
        End If

        ' Scan through all axes in use and set axis parameters
        For nAxis = 0 To gk_nAxesInUse - 1
            nBases(0) = nAxis       ' Set correct (single) axis for Base command
            bOK = axTrioPC.Base(1, nBases)

            ' Modify parameters
            If bOK Then
                bOK = axTrioPC.SetVariable("SERVO", 0#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("ATYPE", 0#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("UNITS", 100#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("SPEED", 5000#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("ACCEL", 3000#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("DECEL", 7000#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("FELIMIT", 10000#)
            End If
            If bOK Then
                bOK = axTrioPC.SetVariable("SERVO", 1#)
            End If
        Next nAxis
        bOK = axTrioPC.SetVariable("WDOG", 1#)  ' Modifications complete so
                                                '   turn watchdog on
    End If
End Sub
```