7 PC MOTION ACTIVEX CONTROL

TrioPC Motion ActiveX Control

The TrioPC ActiveX component provides a direct connection to the Trio MC controllers via a USB or ethernet link. It can be used in any windows programming language supporting ActiveX (OCX) components, such as Visual Basic, Delphi, Visual C, C++ Builder etc.

REQUIREMENTS

- PC with USB and/or ethernet network support
- Windows XP, Windows Vista (32 bit verions) or Windows 7 (32 bit versions)
- Trio PCI driver - for PCI based *Motion Coordinator*s
- Trio USB driver - for *Motion Coordinator* with a USB interface.
- Knowledge of the Trio *Motion Coordinator* to which the TrioPC ActiveX controls will connect.
- Knowledge of the TrioBASIC programming language.

INSTALLATION OF THE ACTIVEX COMPONENT

The component and auxiliary documentation is provided as an MSI installer package. Double clicking on the .msi file will start the install process. It is recommended that any previous version should be uninstalled before the install process is initiated. The installer also installs the Trio USB and Trio PCI drivers and registers the ActiveX component.

USING THE COMPONENT

The TrioPC component must be added to the project within your programming environment. Here is an example using Visual Basic, however the exact sequence will depend on the software package used.

From the Menu select Tools then Choose Toolbox Items.

When the Choose Toolbox Items dialogue box has opened, select the COM components tab, then scroll down until you find “TrioPC Control” then click in the block next to TrioPC. (A tick will appear).

Now click OK and the component should appear in the control panel on the left side of the screen. It is identified as TrioPC Control.

Once you have added the TrioPC component to your form, you are ready to build the project and include the TrioPC methods in your programs.

Connection Commands

Open

DESCRIPTION:
Initialises the connection between the TrioPC ActiveX control and the *Motion Coordinator*.

The connection can be opened over a PCI, Serial, USB or Ethernet link, and can operate in either a synchronous or asynchronous mode. In the synchronous mode all the TrioBASIC methods are available. In the asynchronous mode these methods are not available, instead the user must call SendData() to write to the *Motion Coordinator*, and respond to the OnReceiveChannelx event by calling GetData() to read data received from the *Motion Coordinator*. In this way the user application can respond to asynchronous events which occur on the *Motion Coordinator* without having to poll for them.

If the user application requires the TrioBASIC methods then the synchronous mode should be selected. However, if the prime role of the user application is to respond to events triggered on the *Motion Coordinator*, then the asynchronous method should be used.

SYNTAX:
`Open(PortType, PortMode)`

PARAMETERS:

Short PortType:	See Connection Type.
`Short PortMode:`	See Communications Mode.

RETURN VALUE:
Boolean; **`TRUE`** if the connection is successfully established. For a USB connection, this means the Trio USB driver is active (an MC with a USB interface is on, and the USB connections are correct). If a synchronous connection has been opened the ActiveX control must have also successfully recovered the token list from the *Motion Coordinator*. If the connection is not successfully established this method will return **`FALSE`**.

EXAMPLE:
```
    Rem Open a USB connection and refresh the TriopC indicator
    TrioPC _ Status = TrioPC1.Open(0, 0)
    frmMain.Refresh
```

Close

DESCRIPTION:
Closes the connection between the TrioPC ActiveX control and the *Motion Coordinator*.

SYNTAX:
`Close(PortId)`

PARAMETERS:

Short PortMode:	-1: all ports, 0: synchronous port, >1: asynchronous port
Return Value:	None

EXAMPLE:
```
    Rem Close the connection when form unloads
    Private Sub Form _ Unload(Cancel As Integer)
        TrioPC1.Close
        frmMain.Refresh
    EndSub
```

IsOpen

DESCRIPTION:
Returns the state of the connection between the TrioPC ActiveX control and the *Motion Coordinator*.

SYNTAX:
`IsOpen(PortMode)`

PARAMETERS:

Short PortMode:	See Communications Mode.
Return Value:	Boolean; TRUE if the connection is open, FALSE if it is not .

EXAMPLE:
```
    Rem Close the connection when form unloads
    Private Sub Form _ Unload(Cancel As Integer)
        If TrioPC1.IsOpen(0) Then
            TrioPC1.Close(0)
        End If
        frmMain.Refresh
    End Sub
```

SetHost

DESCRIPTION:
Sets the ethernet host IPV4 address, and must be called prior to opening an ethernet connection. The HostAddress property can also be used for this function

SYNTAX:
`SetHost(host)`

PARAMETERS:

String host:	host IP address as string (eg “192.168.0.250”).
Return Value:	None

EXAMPLE:
```
    Rem Set up the Ethernet IPV4 Address of the target Motion Coordinator
    TrioPC1.SetHost(“192.168.000.001”)
    Rem Open a Synchronous connection
    TrioPC _ Status = TrioPC1.Open(2, 0)
    frmMain.Refresh
```

GetConnectionType

DESCRIPTION
Gets the connection type of the current connection.

SYNTAX:
`GetConnectionType()`

PARAMETERS:
None

RETURN VALUE:
-1: No Connection, See Connection Type.

EXAMPLE:
```
    Rem Open a Synchronous connection
    ConnectError = False
    TrioPC _ Status = TrioPC1.Open(0, 0)
    ConnectionType = TrioPC1.GetConnectionType()
```

```
If ConnectionType <> 0 Then
    ConnectError = True
End If
frmMain.Refresh
```

Properties

Board

DESCRIPTION
Sets the board number used to access a PCI card.

The PCI cards in a PC are always enumerated sequentially starting at 0. It must be set before the **OPEN** command is used.

TYPE:
Long

ACCESSREAD / WRITE

DEFAULT VALUE:
0

EXAMPLE:
```
    Rem Open a PCI connection and refresh the TrioPC indicator
    If TrioPC.Board <> 0 Then
        TrioPC.Board = 0
    End If
    TrioPC _ Status = TrioPC1.Open(3, 0)
    frmMain.Refresh
```

HostAddress

DESCRIPTION:
Used for reading or setting the IPV4 host address used to access a *Motion Coordinator* over an Ethernet connection. The SetHost command can also be used for setting the host adddress.

TYPE:
String

ACCESS:
Read / Write

DEFAULT VALUE:
“192.168.0.250”

EXAMPLE:
```
    Rem Open a Ethernet connection and refresh the TrioPC indicator
    If TrioPC.HostAddress <> “192.168.0.111” Then
        TrioPC.HostAddress = “192.168.0.111”
    End If
    TrioPC _ Status = TrioPC1.Open(2, 0)
    frmMain.Refresh
```

CmdProtocol

DESCRIPTION:
Used to specify the version of the ethernet communications protocol to use to be compatible with the firmware in the ethernet daughterboard. The following values should be used:

0: for ethernet daughterboard firmware version 1.0.4.0 or earlier.

1: for ethernet daughterboard firmware version 1.0.4.1 or later.

TYPE:
Long

ACCESS:
Read / Write

DEFAULT VALUE:
1

EXAMPLE:
```
    Rem Set ethernet protocol for firmware 1.0.4.0
    TrioPC.CmdProtocol = 0
```

Users of older daughterboards will need to update their programs to set the value of this proporty to 0.

FlushBeforeWrite

DESCRIPTION:
The USB and serial communications interfaces are error prone in electrically noisy environments. This means that spurious characters can be received on these interfaces which will cause errors in the OCX. If FlushBeforeWrite is non-zero then the OCX will flush the communications interface before sending a new request, so minimizing the consequences of a noisy environment. The flush routine clears the current contents of the communications buffer and waits 100ms to make sure that there are no other pending characters coming in.

TYPE:
Long

ACCESS:
Read / write

EXAMPLE:
```
    TrioPC1.FlushBeforeWrite = 0
```

FastSerialMode

DESCRIPTION:
The Trio *Motion Coordinator* have two standard RS232 communications modes: slow and fast. The slow mode has parameters 9600,7,e,1 whereas the fast mode has parameters 38400,8,e,1. If FastSerialMode is **FALSE** then the RS232 connection will use the slow mode parameters. If the FastSerialMode is **TRUE** then the RS232 connection will use the fast mode parameters.

ACCESS:
Read / write

TYPE:
Boolean

EXAMPLE:
```
    TrioPC1.FastSerialMode = True
```

Motion Commands

MoveRel

DESCRIPTION
Performs the corresponding **`MOVE`**(...) command on the *Motion Coordinator*.

SYNTAX:
```
MoveRel(Axes, Distance, [Axis])
```

PARAMETERS:

short Axes:	Number of axes involved in the MOVE command.
Double Distance:	Distance to be moved, can be a single numeric value or an array of numeric values that contain at least Axes values.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Base

DESCRIPTION:
Performs the corresponding **`BASE`**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Base(Axes,[Order])
```

PARAMETERS:

short Axes:	Number of axes involved in the move command.
Short Order:	A single numeric value or an array of numeric values that contain at least Axes values that specify the axis ordering for the subsequent motion commands.

RETURN VALUE:
See TrioPC STATUS.

MoveAbs

DESCRIPTION:
Performs the corresponding **MOVEABS**(...) **AXIS**(...) command on the.

SYNTAX:
MoveAbs(Axes, Distance, [Axis])

PARAMETERS:

short Axes:	Number of axes involved in the **MOVEABS** command.
Double Distance:	Absolute position(s) that specify where the move must terminate. This can be a single numeric value or an array of numeric values that contain at least Axes values.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

MoveCirc

DESCRIPTION:
Performs the corresponding **MOVECIRC**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
MoveCirc(EndBase, EndNext, CentreBase, CentreNext, Direction, [Axis])

PARAMETERS:

Double EndBase:	Distance to the end position on the base axis.
Double EndNext:	Distance to the end position on the axis that follows the base axis.
Double CentreBase:	Distance to the centre position on the base axis.
Double CentreNext:	Distance to the centre position on the axis that follows the base axis.
Short Dir:	A numeric value that sets the direction of rotation. A value of 1 implies a clockwise rotation on a positive axis set, 0 implies an anti-clockwise rotation on a positive axis set.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

AddAxis

DESCRIPTION:
Performs the corresponding **`ADDAX`**(...) command on the *Motion Coordinator*.

SYNTAX:
`AddAxis(LinkAxis, [Axis])`

PARAMETERS:

short LinkAxis:	A numeric value that specifies the axis to be “added” to the base axis.
short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

CamBox

DESCRIPTION:
Performs the corresponding **`CAMBOX`**(...) command on the *Motion Coordinator*.

SYNTAX:
`CamBox(TableStart, TableStop, Multiplier, LinkDist, LinkAxis, LinkOption, LinkPos, [Axis])`

PARAMETERS:

Short TableStart:	The position in the table data on the *Motion Coordinator* where the cam pattern starts.
Short TableStop:	The position in the table data on the *Motion Coordinator* where the cam pattern stops.
Double Multiplier:	The scaling factor to be applied to the cam pattern.
Double LinkDist:	The distance the input axis must move for the cam to complete.
Short LinkAxis:	Definition of the Input Axis.

Short LinkOption:	1. link commences exactly when registration event occurs on link axis. 2. link commences at an absolute position on link axis (see param 7). 4. CAMBOX repeats automatically and bi-directionally when this bit is set. 8. Pattern Mode. 32. Link is only active during positive moves.
Double LinkPos:	The absolute position on the link axis where the cam will start.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Cam

DESCRIPTION
Performs the corresponding CAM(...) **`AXIS`**(...) command on the *Motion Coordinator*.

SYNTAX:
`Cam(TableStart, TableStop, Multiplier, LinkDistance, [Axis])`

PARAMETERS:

Short TableStart:	The position in the table data on the *Motion Coordinator* where the cam pattern starts.
Short TableStop:	The position in the table data on the *Motion Coordinator* where the cam pattern stops.
Double Multiplier:	The scaling factor to be applied to the cam pattern.
Double LinkDistance:	Used to calculate the duration in time of the cam. The LinkDistance/Speed on the base axis specifies the duration. The Speed can be modified during the move, and will affect directly the speed with which the cam is performed.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Cancel

DESCRIPTION:
Performs the corresponding **`CANCEL`**(...) **`AXIS`**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Cancel(Mode,[Axis])
```

PARAMETERS:

Short Mode:	Cancel mode. 0 cancels the current move on the base axis. 1 cancels the buffered move on the base axis.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Connect

DESCRIPTION:
Performs the corresponding **CONNECT**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Connect(Ratio, LinkAxis, [Axis])
```

PARAMETERS:

Double Ratio:	The gear ratio to be applied.
Short LinkAxis:	The driving axis.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Datum

DESCRIPTION:
Performs the corresponding **DATUM**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Datum(Sequence, [Axis])
```

PARAMETERS:
The type of datum procedure to be performed:

Short sequence:	0	The current measured position is set as demand position (this is especially useful on stepper axes with position verification). DATUM(0) will also reset a following error condition in the AXISSTATUS register for all axes.
Short Axis:	1	The axis moves at creep speed forward till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
	2	The axis moves at creep speed in reverse till the Z marker is encountered. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error.
	3	The axis moves at the programmed speed forward until the datum switch is reached. The axis then moves backwards at creep speed until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error .
	4	The axis moves at the programmed speed reverse until the datum switch is reached. The axis then moves at creep speed forward until the datum switch is reset. The Demand position is then reset to zero and the Measured position corrected so as to maintain the following error .
	5	The axis moves at programmed speed forward until the datum switch is reached. The axis then moves at creep speed until the datum switch is reset. The axis is then reset as in mode 2.
	6	The axis moves at programmed speed reverse until the datum switch is reached. The axis then moves at creep speed forward until the datum switch is reset. The axis is then reset as in mode 1.
	Optional parameters that must be a single numeric value that specifies the base axis for this move	

RETURN VALUE:
See TrioPC STATUS.

Forward

DESCRIPTION:
Performs the corresponding **FORWARD**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
`Forward([Axis])`

PARAMETER:

Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

Reverse

DESCRIPTION:
Performs the corresponding **REVERSE**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Reverse([Axis])
```

PARAMETERS:

Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

MoveHelical

DESCRIPTION:
Performs the corresponding **MOVEHELICAL**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
```
MoveHelical(FinishBase, FinishNext, CentreBase, CentreNext, Direction,
LinearDistance, [Axis])
```

PARAMETERS:

Double FinishBase:	Distance to the finish position on the base axis.
Double FinishNext:	Distance to the finish position on the axis that follows the base axis.
Double CentreBase:	Distance to the centre position on the base axis.
Double CentreNext:	Distance to the centre position on the axis that follows the base axis.
Short Direction:	A numeric value that sets the direction of rotation. A value of 1 implies a clockwise rotation on a positive axis set, 0 implies an anti-clockwise rotation on a positive axis set.
Double LinearDistance:	The linear distance to be moved on the base axis + 2 whilst the other two axes are performing the circular move.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

MoveLink

DESCRIPTION:
Performs the corresponding **MOVELINK**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
```
MoveLink(Distance, LinkDistance, LinkAcceleration, LinkDeceleration, LinkAxis,
LinkOptions, LinkPosition, [Axis])
```

PARAMETERS:

Double Distance:	Total distance to move on the base axis.
Double LinkDistance:	Distance to be moved on the driving axis.
Double LinkAcceleration	Distance to be moved on the driving axis during the acceleration phase of the move.
Double LinkDeceleration	Distance to be moved on the driving axis during the deceleration phase of the move.
Short LinkAxis:	The driving axis for this move.
Short LinkOptions:	Specifies special processing for this move: 0 no special processing. 1 link commences exactly when registration event occurs on link axis. 2 link commences at an absolute position on link axis (see param 7). 4 MOVELINK repeats automatically and bi-directionally when this bit is set. (This mode can be cleared by setting bit 1 of the REP_OPTION axis parameter). 32 Link is only activee during positive moves on the link axis.
Double LinkPosition:	The absolute position on the link axis where the move will start.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

MoveModify

DESCRIPTION
Performs the corresponding **MOVEMODIFY**(...) **AXIS**(...) command on the *Motion Coordinator*.

SYNTAX:
`MoveModify(Position,[Axis]`

PARAMETERS:

Double Position:	Absolute position of the end of move for the base axis.
Short Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move.

RETURN VALUE:
See TrioPC STATUS.

RapidStop

DESCRIPTION:
Performs the corresponding **`RAPIDSTOP`**(...) command on the *Motion Coordinator*.

PARAMETERS:
None

RETURN VALUE:
See TrioPC STATUS.

Process Control Commands

Run

DESCRIPTION:
Performs the corresponding RUN(...) command on the *Motion Coordinator*.

SYNTAX:
`Run(Program, Process)`

PARAMETERS:

String Program:	String that specifies the name of the program to be run.
Short Process:	Optional parameter that must be a single numeric value that specifies the process on which to run this program.

RETURN VALUE:
See TrioPC STATUS.

Stop

DESCRIPTION:
Performs the corresponding **STOP**(...) command on the *Motion Coordinator*.

SYNTAX:
`Stop(Program, Process)`

PARAMETERS:

String Program:	String that specifies the name of the program to be stopped.
Short Process:	Optional parameter that must be a single numeric value that specifies the process on which the program is running.

RETURN VALUE:
See TrioPC STATUS.

GetTable

DESCRIPTION:
Retrieves and writes the specified table values into the given array.

SYNTAX:
`GetTable(StartPosition, NumberOfValues, Values)`

PARAMETERS

Long StartPosition:	Table location for first value in array.
Long NumberOfValues:	Size of array to be transferred from Table Memory.
Double Values:	A single numeric value or an array of numeric values, of at least size NumberOfValues, into which the values retrieved from the Table Memory will be stored.

RETURN VALUE:
See TrioPC STATUS.

GetVariable

DESCRIPTION:
Returns the current value of the specified system variable. To specify different base axes, the **`BASE`** command must be used.

SYNTAX:
`GetVariable(Variable, Value)`

PARAMETERS:

String Variable:	Name of the system variable to read.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

GetVr

DESCRIPTION:
Returns the current value of the specified VR variable.

SYNTAX:
`GetVr(Variable, Value)`

PARAMETERS:

Short Variable:	Number of the VR variable to read.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

SetTable

DESCRIPTION:
Sets the specified table variables to the values given in an array.

SYNTAX:
`SetTable(StartPosition, NumberOfValues, Values)`

PARAMETERS

Long StartPosition:	Table location for first value in array.
Long NumberOfValues:	Size of array to be transferred to Table Memory.
Double Values:	A single numeric value or an array of numeric values that contain at least NumberOfValues values to be placed in the Table Memory.

RETURN VALUE:
See TrioPC STATUS.

SetVariable

DESCRIPTION:
Sets the current value of the specified system variable. To specify different base axes, the **`BASE`** command

must be used.

SYNTAX:
`SetVariable(Variable, Value)`

PARAMETERS:

String Variable:	Name of the system variable to write.
Double Value:	Variable in which the value to write is stored.

RETURN VALUE:
See TrioPC STATUS.

SetVr

DESCRIPTION:
Sets the value of the specified Global variable.

SYNTAX:
`SetVr(Variable, Value)`

PARAMETERS:

Short Variable:	Number of the VR variable to write.
Double Value:	Variable in which the value to write is stored.

RETURN VALUE:
See TrioPC STATUS.

GetProcessVariable

DESCRIPTION:
Returns the current value of a variable from a currently running process. It is quite difficult to calculate the VariableIndex as the storage for the named variables is assigned during the program compilation, but it is not stored due to memory restrictions on the *Motion Coordinator*s. To make things worse, if a program is modified in such a way the named variables it uses are changed (added, removed, or changed in order of use) then the indices may change.

SYNTAX:

```
GetProcessVariable(VariableIndex, Process, Value)
```

PARAMETERS:

Short VariableIndex:	The index of the variable in the process variables table.
Short Process:	The process number of the running process.
Double Value:	Variable in which to store the value read.

EXAMPLE:

Let us assume that there is the program “T1” on the *Motion Coordinator* which has the following contents:

```
    y=2
    x=1
```

If this program is run on process 1 by the command RUN “T1”,1 then we could use the following code in VisualBASIC to read the contents of the x and y variables.

```
    Dim x As Double
    Dim y As Double
    If Not AxTrioPC1.GetProcessVariable(1, 1, x) Then Exit Sub
    If Not AxTrioPC1.GetProcessVariable(0, 1, y) Then Exit Sub
    MsgBox(“X has value “ + Format(x))
    MsgBox(“Y has value “ + Format(y))
```

RETURN VALUE:

See TrioPC STATUS.

GetAxisVariable

DESCRIPTION:

For a system variable that accepts the **AXIS** modifier this method will return the value of the that system variable on the given axis. If the system variable does not exist, or does not accept the **AXIS** modifier, then this method will fail.

SYNTAX:

```
GetAxisVariable(VariableIndex, Axis, Value)
```

PARAMETERS:

String Variable:	The name of the variable.
Short Axis:	The axis number.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

SetAxisVariable

DESCRIPTION:
For a system variable that accepts the **`AXIS`** modifier this method will set the value of the that system variable on the given axis. If the system variable does not exist, or does not accept the **`AXIS`** modifier, then this method will fail.

SYNTAX:
```
SetAxisVariable(VariableIndex, Axis, Value)
```

PARAMETERS:

String Variable:	The name of the variable.
Short Axis:	The axis number.
Double Value:	Value to set.

RETURN VALUE:
See TrioPC STATUS.

GetProcVariable

DESCRIPTION:
For a system variable that accepts the **`PROC`** modifier this method will return the value of the that system variable on the given process. If the system variable does not exist, or does not accept the **`PROC`** modifier, then this method will fail.

SYNTAX:
```
GetProcVariable(Variable, Process, Value)
```

PARAMETERS:

String Variable:	The name of the variable.
Short Process:	The process number of the running process.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

SetProcVariable

DESCRIPTION:
For a system variable that accepts the **`PROC`** modifier this method will set the value of the that system variable on the given process. If the system variable does not exist, or does not accept the **`PROC`** modifier, then this method will fail.

SYNTAX:
`SetProcVariable(Variable, Process, Value)`

PARAMETERS:

String Variable:	The name of the variable.
Short Process:	The process number of the running process.
Double Value:	Value to set.

RETURN VALUE:
See TrioPC STATUS.

GetSlotVariable

DESCRIPTION:
For a system variable that accepts the **`SLOT`** modifier this method will return the value of the that system variable on the given slot. If the system variable does not exist, or does not accept the **`SLOT`** modifier, then this method will fail.

SYNTAX:
`GetSlotVariable(Variable, Slot, Value)`

PARAMETERS:

String Variable:	The name of the variable.
Short Slot:	The slot number.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

SetSlotVariable

DESCRIPTION:
For a system variable that accepts the **`SLOT`** modifier this method will set the value of the that system variable on the given slot. If the system variable does not exist, or does not accept the **`SLOT`** modifier, then this method will fail.

SYNTAX:
`SetSlotVariable(Variable, Slot, Value)`

PARAMETERS:

String Variable:	The name of the variable.
Short Slot:	The slot number.
Double Value:	Value to set.

RETURN VALUE:
See TrioPC STATUS.

GetPortVariable

DESCRIPTION:
For a system variable that accepts the **`PORT`** modifier this method will return the value of the that system variable on the given port. If the system variable does not exist, or does not accept the **`PORT`** modifier, then this method will fail.

SYNTAX:
`GetPortVariable(Variable, Port, Value)`

PARAMETERS:

String Variable:	The name of the variable.
Short Port:	The port number.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

SetPortVariable

DESCRIPTION:
For a system variable that accepts the **`PORT`** modifier this method will set the value of the that system variable on the given port. If the system variable does not exist, or does not accept the **`PORT`** modifier, then this method will fail.

SYNTAX:
`SetPortVariable(Variable, Port, Value)`

PARAMETERS:

String Variable:	The name of the variable.
Short Port:	The port number.
Double Value:	Value to set.

RETURN VALUE:
See TrioPC STATUS.

Input / Output Commands

Ain

DESCRIPTION:
Performs the corresponding AIN(...) command on the *Motion Coordinator*.

SYNTAX:
```
Ain(Channel, Value)
```

PARAMETERS:

Short Channel:	AIN channel to be read.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

Get

DESCRIPTION:
Performs the corresponding GET #... command on the *Motion Coordinator*.

SYNTAX:
```
Get(Channel, Value)
```

PARAMETERS:

Short Channel:	Comms channel to be read.
Short Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

In

DESCRIPTION:
Performs the corresponding IN(...) command on the *Motion Coordinator*.

SYNTAX:
`In(StartChannel, StopChannel, Value)`

PARAMETERS:

Short StartChannel:	First digital I/O channel to be checked.
Short StopChannel:	Last digital I/O channel to be checked.
Long Value:	Variable to store the value read.

RETURN VALUE:
See TrioPC STATUS.

Input

DESCRIPTION:
Performs the corresponding **`INPUT`** #... command on the *Motion Coordinator*.

SYNTAX:
`Input(Channel, Value)`

PARAMETERS:

Short Channel:	Comms channel to be read.
Double Value:	Variable in which to store the value read.

RETURN VALUE:
See TrioPC STATUS.

Key

DESCRIPTION:
Performs the corresponding KEY #... command on the *Motion Coordinator*.

SYNTAX:

`Key(Channel, Value)`

PARAMETERS:

Short Channel:	Comms channel to be read.
Double Value:	Variable in which to store the value read.

RETURN VALUE:

See TrioPC STATUS.

Linput

DESCRIPTION:

Performs the corresponding **`LINPUT`** # command on the *Motion Coordinator*.

SYNTAX:

`Linput(Channel, Startvr)`

PARAMETERS:

Short Channel:	Comms channel to be read.
Short StartVr:	Number of the VR variable into which to store the first key press read.

RETURN VALUE:

See TrioPC STATUS.

Mark

DESCRIPTION:

Performs the corresponding **`MARK`**(...) command on the *Motion Coordinator*.

SYNTAX:

`Mark(Axis, Value)`

PARAMETERS:

Short Axis number:	Axis number.
Short Value:	The stored capture value for a registration first event.

RETURN VALUE:
See TrioPC STATUS. **`FALSE`** if no value has been captured (no registration first event has occurred).

MarkB

DESCRIPTION:
Performs the corresponding **`MARKB`**(...) command on the *Motion Coordinator*.

SYNTAX:
`MarkB(Axis, Value)`

PARAMETERS:

Short Axis number:	Axis number.
Short Value:	The stored capture value for a registration second event.

RETURN VALUE:
See TrioPC STATUS. **`FALSE`** if no value has been captured (no registration second event has occurred).

Op

DESCRIPTION:
Performs the corresponding OP(...) command on the *Motion Coordinator*.

SYNTAX:
`Op(Output, [State])`

PARAMETERS:

Long Output:	Numeric value. If this is the only value specified then it is the bit map of the outputs to be specified, otherwise it is the number of the output to be written.
Short State:	Optional numeric value that specifies the desired status of the output, 0 implies off, not-0 implies on.

RETURN VALUE:
See TrioPC STATUS.

Pswitch

DESCRIPTION:
Performs the corresponding **`PSWITCH`**(…) command on the *Motion Coordinator*.

SYNTAX:
```
Pswitch(Switch, Enable, Axis, OutputNumber, OutputStatus, SetPosition,
ResetPosition)
```

PARAMETERS:

Short Switch:	Switch to be set.
Short Enable:	1 to enable, 0 to disable.
Short Axis:	Optional numeric value that specifies the base axis for this command.
Short OutputNumber:	Optional numeric value that specifies the number of the output to set.
Short OutputStatus:	Optional numeric value that specifies the signalled status of the output, 0 implies off, not-0 implies on.
Double SetPosition:	Optional numeric value that specifies the position at which to signal the output.
Double ResetPosition:	Optional numeric value that specifies the position at which to reset the output.

RETURN VALUE:
See TrioPC STATUS.

ReadPacket

DESCRIPTION:
Performs the corresponding **`READPACKET`**(…) command on the *Motion Coordinator*.

SYNTAX:
```
ReadPacket(PortNumber, StartVr, NumberVr, Format)
```

PARAMETERS:

Short PortNumber:	Number of the comms port to read (0 or 1).
Short StartVr:	Number of the first variable to receive values read from the comms port.
Short NumberVr:	Number of variables to receive.
Short Format:	Numeric format in which the numbers will arrive.

RETURN VALUE:
See TrioPC STATUS.

Record

DESCRIPTION:
This method is no longer supported by any current *Motion Coordinator*.

Regist

DESCRIPTION:
Performs the corresponding **`REGIST`**(...) command on the *Motion Coordinator*.

SYNTAX:
```
Regist(Mode, Dist)
```

PARAMETERS:

Short Mode:	Registration mode. 1. Axis absolute position when Z Mark Rising. 2. Axis absolute position when Z Mark Falling. 3. Axis absolute position when Registration Input Rising. 4. Axis absolute position when Registration Input Falling. 5. Unused. 6. R input rising into REG_POS and Z mark rising into REG_POSB. 7. R input rising into REG_POS and Z mark falling into REG_POSB. 8. R input falling into REG_POS and Z mark rising into REG_POSB. 9. R input falling into REG_POS and Z mark falling into REG_POSB.
Double Dist:	Only used in pattern recognition mode and specifies the distance over which to record the transitions.

RETURN VALUE:
See TrioPC STATUS.

Send

DESCRIPTION:
Performs the corresponding **`SEND`**(...) command on the *Motion Coordinator*.

SYNTAX:

`Send(Destination, Type, Data1, Data2)`

PARAMETERS:

Short Destination:	Address to which the data will be sent.
Short Type:	type of message to be sent: 1 . Direct variable transfer. 2 . Keypad offset.
Short Data1:	Data to be sent. If this is a keypad offset message then it is the offset, otherwise it is the number of the variable on the remote node to be set.
Short Data2:	Optional numeric value that specifies the value to be set for the variable on the remote node.

RETURN VALUE:

See TrioPC STATUS.

Setcom

DESCRIPTION:

Performs the corresponding **SETCOM**(...) command on the *Motion Coordinator*.

SYNTAX:

`Setcom(Baudrate, DataBits, StopBits, Parity, [Port], [Control])`

PARAMETERS:

Long BaudRate:	Baud rate to be set.
Short DataBits:	Number of bits per character transferred.
Short StopBits:	Number of stop bits at the end of each character.
Short Parity:	Parity mode of the port (0=>none, 1=>odd, 2=> even).
Short Port:	Optional numeric value that specifies the port to set (0..3).
Short Control:	Optional numeric value that specifies whether to enable or disable handshaking on this port.

RETURN VALUE:

See TrioPC STATUS.

General commands

Execute

DESCRIPTION:
Performs the corresponding **`EXECUTE`**... command on the *Motion Coordinator*.

SYNTAX:
`Execute(Command)`

PARAMETERS:
String Command: String that contains a valid TrioBASIC command.

RETURN VALUE:
Boolean; **`TRUE`** if the command was sent successfully to the *Motion Coordinator* and the **`EXECUTE`** command on the *Motion Coordinator* was completed successfully and the command specified by the **`EXECUTE`** command was tokenised, parsed and completed successfully. Otherwise **`FALSE`**.

GetData

DESCRIPTION:
This method is used when an asynchronous connection has been opened, to read data received from the *Motion Coordinator* over a particular channel. The call will empty the appropriate channel receive data buffer held by the ActiveX control.

SYNTAX:
`GetData(channel, data)`

PARAMETERS:

Short channel:	Channel over which the required data was received (0,5,6,7, or 9).
String data:	data received by the control from the *Motion Coordinator*.

RETURN VALUE:
Boolean; **`TRUE`** - if the given channel is valid, the connection open and the data read correctly from the buffer. Otherwise **`FALSE`**.

SendData

DESCRIPTION

This method is used when the connection has been opened in the asynchronous mode, to write data to the *Motion Coordinator* over a particular channel.

SYNTAX:

```
SendData(channel, data)
```

PARAMETERS:

Short channel:	channel over which to send the data (0,5,6,7, or 9).
String data:	data to be written to the *Motion Coordinator*.

RETURN VALUE:

Boolean; **TRUE** - if the given channel is valid, the connection open, and the data written out correctly. Otherwise **FALSE**.

Scope

DESCRIPTION:

Initialises the data capture system in the *Motion Coordinator* for future data capture on a trigger event by executing a **SCOPE** command on the *Motion Coordinator*. A trigger event occurrs when the *Motion Coordinator* executes a **TRIGGER** command.

SYNTAX:

```
Scope(OnOff, [SamplePeriod, TableStart, TableEnd, CaptureParams])
```

PARAMETERS:

Boolean OnOff:	TRUE to set up and enable data capture, FALSE to disable it.
Long SamplePeriod:	Data sample period (in servo periods).
Long TableStart:	The table index for the start of the block of TABLE memory which will be used to hold captured data.
Long TableEnd:	The table index for the start of the block of TABLE memory which will be used to hold captured data.
String CaptureParams:	A string of up to 4 comma seperated parameters to capture.

EXAMPLE:

```
    Rem Set up to capture MPOS and DOPS on axis 5
    TrioPC _ Status = TrioPC1.Scope(True, 10, 0, 1000, “MPOS AXIS(5), DPOS
    AXIS(5)"")
```

RETURN VALUE:
See TrioPC STATUS.

Trigger

DESCRIPTION:
Sends a **`TRIGGER`** command to the *Motion Coordinator* to start data capture previously configured using a **`SCOPE`** command.

SYNTAX:
```
Trigger()
```

PARAMETERS:
None.

RETURN VALUE:
See TrioPC STATUS.

Events

OnBufferOverrunChannel0/5/6/7/9

DESCRIPTION:
One of these events will fire if a particular channel data buffer overflows. The ActiveX control stores all data received from the *Motion Coordinator* in the appropriate channel buffer when the connection has been opened in asynchronous mode. As data is received it is the responsibility of the user application to call the GetData() method whenever the OnReceiveChannelx event fires (or otherwise to call the method periodically) to prevent a buffer overrun. Which event is fired will depend upon which channel buffer overran.

SYNTAX:
```
OnBufferOverrunChannelx()
```
The channel number (x) can be any of the following: 0, 5, 6, 7 or 9.

PARAMETERS:
None.

RETURN VALUE:
None.

OnReceiveChannel0/5/6/7/9

DESCRIPTION:
One of these events will fire when data is received from the *Motion Coordinator* over a connection which has been opened in the asynchronous mode. Which event is fired will depend upon over which channel the *Motion Coordinator* sent the data. It is the responsibility of the user application to call the GetData() method to retrieve the data received.

SYNTAX:
```
OnReceiveChannelx()
```
The channel number (x) can be any of the following: 0, 5, 6, 7 or 9.

PARAMETERS:
None.

RETURN VALUE:
None.

OnProgress

DESCRIPTION:
The file operations LoadProgram, LoadProject and LoadSystem can take a long time to complete. To give some feedback on this process the OnProgress event is fired periodically during the file operation.

SYNTAX:
```
OnOnProgress
```

PARAMETERS:

Description:	Textual description of the associated process
Percentage:	Progress of the process in percent.

MechatroLink

DESCRIPTION:
Performs the corresponding **`MECHATROLINK`**(...) command on the *Motion Coordinator*. For more information on the **`MECHATROLINK`** command please see the corresponding *Motion Coordinator* user manual. This method will only work on those *Motion Coordinator*s that support the MehchatroLink interface.

SYNTAX:
```
MechatroLink(Module, Function, NumberOfParameters, MLParameters, Result)
```

PARAMETERS:

Short Module:	Number of the MechatroLink interface module.
Short Function:	MechatroLink function number.
Short NumberOfParameters:	Number of parameters to use in the MECHATROLINK command.
Double MLParameters:	Array of parameters to use for the MECHATROLINK command.
Double Result:	Variable in which the return value is stored.

RETURN VALUE:
See TrioPC STATUS.

Program Manipulation Commands

LoadProject

DESCRIPTION:
Not implemented.

LoadSystem

DESCRIPTION:
Not implemented.

LoadProgram

DESCRIPTION:
Not implemented.

New

DESCRIPTION:
Deletes a program on the *Motion Coordinator*.

SYNTAX:
`New(Program)`

PARAMETERS:

String Program:	The name of the program to be deleted.

RETURN VALUE:
See TrioPC STATUS.

Select

DESCRIPTION:
Selects a program on the *Motion Coordinator*.

SYNTAX:
```
Select(Program)
```

PARAMETERS:

String Program:	The name of the program to be selected.

RETURN VALUE:
See TrioPC STATUS.

Dir

DESCRIPTION:
Gets a directory listing from the *Motion Coordinator*.

SYNTAX:
```
Dir(Directory)
```

PARAMETERS:

String Program:	A string object used to return the directory listing.

RETURN VALUE:
See TrioPC STATUS.

InsertLine

DESCRIPTION:
Inserts a line into a program onto the *Motion Coordinator*. This will first Select the given program on the controller and then insert the line text at the given line number.

SYNTAX:
```
InsertLine(Program, Line, LineText)
```

PARAMETERS:

String Program:	The name of the program.
Short Line:	The line number at which the new line will be inserted.
String LineText:	The text of the line to be inserted.

RETURN VALUE:
See TrioPC STATUS.

Data Types

The following data types are used by the PC Motion control interface:

Connection Type

ALSO KNOWN AS:
Port Type.

DESCRIPTION:
An enumeration representing communication port type.

Values:	-1:	No connection .
	0:	USB.
	1:	Serial.
	2:	Ethernet.
	3:	PCI.
	4:	Path.
	5:	FINS (Not used on Trio controllers).

Communications Mode

ALSO KNOWN AS:
Port Mode.

DESCRIPTION:
An enumeration representing the operating mode of a communications link.

VALUES:

Interface	Mode	Description
USB:	0	Synchronous.
	1	Asynchronous.
Serial:	>0	Synchronous on specified port number.
	<0	Asynchronous on specified port number.
Ethernet:	0 3240	Synchronous on specified port number.
	23	Asynchronous on specified port number (default 23).

	other	
PCI:	0	Synchronous.
	1	Asynchronous.

TrioPC status

Many of the methods implemented by the TrioPC interface return a boolean status value. The value will be **TRUE** if the command was sent successfully to the *Motion Coordinator* and the command on the *Motion Coordinator* was completed successfully. It will be **FALSE** if it was not processed correctly, or there was a communications error.