CHAPTER

12

USING THE PC MOTION ACTIVEX CONTROL

Introduction

The TrioPC ActiveX component provides a direct connection to the Trio MC controllers via a PCI bus, USB, serial 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 one or more of USB interface, Ethernet network interface, serial port or PCI based *Motion Coordinator.*
- Windows 2000, XP or Vista
- TrioUSB driver - for USB connection
- Trio PCI driver - for PCI connection
- TrioPC OCX
- Knowledge of the Trio *Motion Coordinator* to which the TrioPC ActiveX controls will connect.
- Knowledge of the Trio BASIC programming language.

Installation of the ActiveX Component

Launch the program "Install_TrioPCMotion" and follow the on-screen instructions. The TrioUSB driver and TrioPC ocx will be installed and registered to your Windows environment. The Trio PCI driver will also be installed on systems running Windows 2000 or Windows XP. A Windows Help file is included as an alternative to the printed pages in this manual.

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 Tolbox Items.

When the Choose Tolbox Items dialogue 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 toolbox on the left side of the screen. It is identified as TrioPC Control.

Now add 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 Trio BASIC 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 them.

If the user application requires the Trio BASIC 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:` 0: USB, 1: Serial Port 2: Ethernet 3: PC

`short PortMode:`

USB: 0: Synchronous Mode 1: Asynchronous mode

Serial: >0 Opens synchronous connection on specified port number

Serial: <0 Opens asynchronous connection on specified port number

Ethernet: 0, 3240: Synchronous 23: Asynchronous default port Other: Asynchronous custom port

PCI : 0: Synchronous Mode 1: Asynchronous mode 0: Synchronous Mode 1: Asynchronous mode

Return Value: **TRUE** if the connection is successfully established. For a USB connection, this means the TrioUSB driver is active (an MC with a USB card 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**.Short PortType : 0: USB, 1: Serial Port 2: Ethernet 3: PCI

Short PortMode:

USB: 0: Synchronous Mode 1: Asynchronous mode

Serial: >0 Opens synchronous connection on specified port number

<0 Opens asynchronous connection on specified port number

Ethernet: 0, 3240: Synchronous 23: Asynchronous default port Other: Asynchronous custom port

PCI : 0: Synchronous Mode 1: Asynchronous mode

Example

```
Rem Open a USB connection and refresh the TrioPC indicator
TrioPC_Status = TrioPC1.Open(0, 0)
frmMain.Refresh
```

Note: When **PortType** is set to 1, serial port, then only the syncronous mode is available. I.e. **PortMode** must be set to 0. The serial port must be configured by a program on the controller before the PCMotion component can connect.

```
SETCOM(38400,8,1,2,1,8)
REMOTE(0)
```

Close

Description Closes the connection between the TrioPC ActiveX control and the *Motion Coordinator*

Syntax: **Close(PortMode)**

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(0)
    frmMain.Refresh
End Sub
```

IsOpen

Description Returns the state of the connection between the TrioPC ActiveX control and the *Motion Coordinator*

Syntax: **IsOpen(PortMode)**

Parameters **short PortMode:** -1: all ports, 0: synchronous port, >1: asynchronous port

Return Value: **TRUE** if port is open, **FALSE** if it is closed.

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 IP 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 `VARIANT host:` host IP address (eg 192.168.0.250).

Return Value: `None`

Example

```
Rem Set up the Ethernet IP 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, 0: USB, 1:N/A, 2: Ethernet, 3: PCI

Example

```
Rem Open a Synchronous connection
ConnectError = False
TrioPC_Status = TrioPC1.Open(0, 0)
ConnectionType = TrioPC1.GetConnectionType()
If ConnectionType <> 0 Then
    ConnectError = True
frmMain.Refresh
```

Properties

Board

Description Specifies the board number for a PCI connection. It must be specified before the **OPEN** command is used.

Type Long

Access Read / 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 changing the ethernet host IP address, and must be set prior to opening 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
```

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

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
VARIANT Distance:	Distance to be moved, can be a single numeric value or an array of numeric values that contain at least Axes values
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **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
VARIANT 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: **TrioPC STATUS**.

MoveAbs

Description: Performs the corresponding **MOVEABS(…) AXIS(…)** command on the *Motion Coordinator*

Syntax: `MoveAbs(Axes, Distance, [Axis])`

Parameters:

`short Axes:`	Number of axes involved in the moveabs command
`VARIANT Distance:`	Absolute positions that specify where the move must terminate, can be a single numeric value or an array of numeric values that contain at least Axes values
`VARIANT Axis:`	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: `TrioPC STATUS`.

MoveCirc

Description: Performs the corresponding **`MOVECIRC(…) AXIS(…)`** command on the *Motion Coordinator*

Syntax: `MoveCirc(EndBase, EndNext, CentreBase, CentreNext, Dir, [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.
`VARIANT Axis:`	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: `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.

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

Return Value: **TrioPC STATUS**.

CamBox

Description: Performs the corresponding **CAMBOX(…)** command on the *Motion Coordinator*

Syntax: **CamBox(TableStart, TableStop, Multiplier, LinkDist, LinkAxis, LinkOpt, 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 LinkOpt:

- **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.

double LinkPos: The absolute position on the link axis where the cam will start.

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

Return Value: **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
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **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
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **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
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **TrioPC STATUS**.

Datum

Description: Performs the corresponding **DATUM(…) AXIS(…)** command on the *Motion Coordinator*

Syntax: **Datum(Sequence, [Axis])**

Parameters: **short Sequence:** The type of datum procedure to be performed:

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.
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.

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

Return Value: **TrioPC STATUS**.

Forward

Description Performs the corresponding F**ORWARD(…) AXIS(…)** command on the *Motion Coordinator*

Syntax: **Forward([Axis])**

Parameters **VARIANT Axis:** Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **TrioPC STATUS**.

Reverse

Description: Performs the corresponding **REVERSE(…) AXIS(…)** command on the *Motion Coordinator*

Syntax: **Reverse([Axis])**

Parameters: **VARIANT Axis:** Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **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
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **TrioPC STATUS**.

MoveLink

Description: Performs the corresponding **MOVELINK(…) AXIS(…)** command on the *Motion Coordinator*

Syntax: **MoveLink(Distance, LinkDistance, LinkAcc, LinkDec, LinkAxis, LinkOptions, LinkPosn, [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)
double LinkPosition:	The absolute position on the link axis where the move will start.
VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **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.
	VARIANT Axis:	Optional parameters that must be a single numeric value that specifies the base axis for this move

Return Value: **TrioPC STATUS**.

RapidStop

Description: Performs the corresponding **RAPIDSTOP(…)** command on the *Motion Coordinator*

Parameters: None

Return Value: **TrioPC STATUS**.

Process Control Commands

Run

Description: Performs the corresponding **RUN(…)** command on the *Motion Coordinator*

Syntax: **Run(Program, Process)**

Parameters:

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

Return Value: **TrioPC STATUS**.

Stop

Description: Performs the corresponding **STOP(…)** command on the *Motion Coordinator*

Syntax: **Stop(Program, Process)**

Parameters:

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

Return Value: **TrioPC STATUS**.

Variable Commands

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.
`VARIANT 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: **`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

`BSTR Variable:`	Name of the system variable to read
`double *Value:`	Variable in which to store the value read

Return Value: **`TrioPC STATUS`**.

GetVr

Description: Returns the current value of the specified Global 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: **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.
VARIANT 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: **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

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

Return Value: **TrioPC STATUS**.

SetVr

Description: Sets the value of the specified Global variable.

Syntax: `SetVr(Variable, Value)`

Parameters:

`BSTR Variable:`	Number of the VR variable to write
`double Value:`	Variable in which the value to write is stored.

Return Value: `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: **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
double *Value:	Variable in which to store the value read.

Return Value: **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: **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: **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: **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: **TrioPC STATUS**.

Op

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

Syntax: **Op(Output, State)**

Parameters:	**VARIANT 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.
	VARIANT State:	Optional numeric value that specifies the desired status of the output, 0 implies off, not-0 implies on.

Return Value: **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
VARIANT Axis:	Optional numeric value that specifies the base axis for this command
VARIANT OutputNumber:	Optional numeric value that specifies the number of the output to set
VARIANT OutputStatus:	Optional numeric value that specifies the signalled status of the output, 0 implies off, not-0 implies on.
VARIANT SetPosition:	Optional numeric value that specifies the position at which to signal the output
VARIANT ResetPosition:	Optional numeric value that specifies the position at which to reset the output.

Return Value: **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: **TrioPC STATUS**.

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

double Dist: Only used in pattern recognition mode and specifies the distance over which to record the transitions.

Return Value: **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: **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)
VARIANT Port:	Optional numeric value that specifies the port to set (0..3)
VARIANT Control:	Optional numeric value that specifies whether to enable or disable handshaking on this port

Return Value: **TrioPC STATUS**.

General commands

Execute

Description: Performs the corresponding **EXECUTE …** command on the *Motion Coordinator*.

Syntax: **Execute(Command)**

Parameters **BSTR Command:** String that contains a valid Trio BASIC command

Return Value: **TrioPC STATUS: 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.

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 (5,6,7, or 9).
VARIANT data:	data received by the control from the *Motion Coordinator*

Return Value: **TrioPC STATUS:** TRUE - if the given channel is valid, the connection open and the data read correctly from the buffer.

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 (5,6,7, or 9).

`VARIANT data:` data to be written to the *Motion Coordinator*

Return Value: **`TrioPC STATUS:`** TRUE - if the given channel is valid, the connection open, and the data written out correctly.

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()**

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()**

Parameters: **None**.

Return Value: **None**.

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.